Accelerating Machine Learning on Emerging Architectures

Big Simulation and Big Data Workshop

January 9, 2017
Indiana University

Judy Qiu
Associate Professor of Intelligent Systems Engineering
Indiana University
Outline

1. Motivation: Machine Learning Applications
2. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
3. Harp-DAAL Framework: Design and Implementations
4. Conclusions and Future Work
Acknowledgements

Bingjing Zhang     | Yining Wang     | Langshi Chen    | Meng Li     | Bo Peng     | Yiming Zou

SALSA HPC Group
School of Informatics and Computing
Indiana University

Rutgers University
Virginia Tech
Kansas University
Arizona State University
State University of New York at Stony Brooke
University of Utah

Digital Science Center
Indiana University

Intel Parallel Computing Center
IPCC

NSF
Indiana University
Intel
DATANET: CIF21 DIBBS: Middleware and High Performance Analytics Libraries for Scalable Data Science
Motivation

• Machine learning is widely used in data analytics

• Need for high performance
  – Big data & Big model
  – "Select model and hyper parameter tuning" step need to run the training algorithm for many times

• Key: optimize for efficiency
  – What is the 'kernel' of training?
  – Computation model
Recommendation Engine

- Show us products typically purchased together
- Curate books and music for us based on our preferences
- Have proven significant because they consistently boost sales as well as customer satisfaction
Fraud Detection

- Identify fraudulent activity
- Predict it before it has occurred saving financial services firms millions in lost revenue.
- Analysis of financial transactions, email, customer relationships and communications can help
More Opportunities...

• Predicting customer “churn” – when a customer will leave a provider of a product or service in favor of another.

• Predicting presidential elections, whether a swing voter would be persuaded by campaign contact.

• Google has announced that it has used Deep Mind to reduce the energy used for cooling its datacenter by 40 per cent.

• Imagine...
The Process of Data Analytics

• Define the Problem
  – Binary or multiclass, classification or regression, evaluation metric, ...

• Dataset Preparation
  – Data collection, data munging, cleaning, split, normalization, ...

• Feature Engineering
  – Feature selection, dimension reduction, ...

• Select model and hyper parameter tuning
  – Random Forest, GBM, Logistic Regression, SVM, KNN, Ridge, Lasso, SVR, Matrix Factorization, Neural Networks, ...

• Output the best models with optimized hyper parameters
Challenges from Machine Learning Algorithms

Machine Learning algorithms in various domains:
• Biomolecular Simulations
• Epidemiology
• Computer Vision

They have:
• Iterative computation workload
• High volume of training & model data

Traditional Hadoop/MapReduce solutions:
• Low Computation speed (lack of multi-threading)
• High data transfer overhead (disk based)
### Taxonomy for ML Algorithms

<table>
<thead>
<tr>
<th>Task Level</th>
<th>Classification</th>
<th>Clustering</th>
<th>Regression</th>
<th>Recommendation</th>
<th>Structure Learning</th>
<th>Dimensionality Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling Level</td>
<td>General Linear Model</td>
<td>Kernel Method</td>
<td>Nearest Neighbor</td>
<td>Decision Tree</td>
<td>Factorization Machine</td>
<td>Graphical Model</td>
</tr>
<tr>
<td>Solver Level</td>
<td>SVD, PCA, QR, Eigen, ALS…</td>
<td>GD, SGD, LBFGS, CG, CCD…</td>
<td>Linear Algebra Kernel</td>
<td>Numerical Optimization</td>
<td>EM: EM, VB; BP; MCMC: Gibbs Sampling, Metropolis-Hastings…</td>
<td>Statistical Inference</td>
</tr>
</tbody>
</table>

- Task level: describe functionality of the algorithm
- Modeling level: the form and structure of model
- Solver level: the computation pattern of training
Outline

1. Motivation: Machine Learning Applications
2. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
3. Harp-DAAL Framework: Design and Implementations
4. Conclusions and Future Work
Emerging Many-core Platforms

Comparison of Many-core and Multi-core Architectures

• Much more number of cores
• Lower single core frequency
• Higher data throughput

How to explore computation and Bandwidth of KNL for Machine Learning applications?
Intel Xeon/Haswell Architecture

- Much more number of cores
- Lower single core frequency
- Higher data throughput
**Intel Xeon Phi (Knights Landing) Architecture**

- Up to 72 cores, 288 threads connected in a 2D-mesh
- High bandwidth (> 400 GB/s) Memory (MCDRAM)
- Up to 144 AVX512 vectorization units (VPUs)
- 3 Tflops (DP) performance delivery
- Omni-path link among processors (~ 100 GB/s)
DAAL: Intel’s Data Analytics Acceleration Library

DAAL is an open-source project that provides:

- Algorithms Kernels to Users
  - Batch Mode (Single Node)
  - **Distributed Mode (multi nodes)**
  - Streaming Mode (single node)
- Data Management & APIs to Developers
  - Data structure, e.g., Table, Map, etc.
  - HPC Kernels and Tools: MKL, TBB, etc.
  - Hardware Support: Compiler
Case Study: Matrix-Factorization Based on SGD (MF-SGD)

\[ X = UV \]

Decompose a large matrix into two model matrices, used in Recommender systems

- Large Training Data: Tens of millions of points
- Large Model Data: m, n could be millions
- Random Memory Access Pattern in Training
Stochastic Gradient Descent

The standard SGD will loop over all the nonzero ratings $x_{i,j}$ in a random way

- Compute the errors
  \[ e_{i,j} = x_{i,j} - u_{i,*}v_{*,j} \]

- Update the factors $U$ and $V$
  \[ u_{i,*} = u_{i,*} + \gamma \cdot (e_{i,j} \cdot v_{*,j} - \lambda \cdot u_{i,*}) \]
  \[ v_{*,j} = v_{*,j} + \gamma \cdot (e_{i,j} \cdot u_{i,*} - \lambda \cdot v_{*,j}) \]
Challenge of SGD in Big Model Problem

1. Memory Wall
   - 4 memory ops for 3 computation ops
   - In updating U and V
   
   Processor is hungry of data!!

2. Random Memory Access
   - Difficulty in data prefetching
   - Inefficiency in using cache

We test a multi-threading SGD on a CPU

The strong scalability collapses after using more than 16 threads!!
What for Novel Hardware Architectures and Runtime Systems

**Hardware Aspect:**
- 3D stack memory
- Many-core: GPU, Xeon Phi, FPGA, etc.

**Software Aspect:**
- Runtime System
- Dynamic Task Scheduling

*Reduce the memory access latency*
*Increase memory bandwidth*

*IBM and Micron’s big memory cube*

*A generalized architecture for an FPGA*
Intra-node Performance: DAAL-MF-SGD vs. LIBMF

Test on KNL with Yahoomusic Dataset

LIBMF: a start-of-art open source MF-SGD package
- Only single node mode
- Highly optimized for memory usage

We compare our DAAL-MF-SGD kernel with LIBMF on a single KNL node, using YahooMusic dataset
DAAL-MF-SGD delivers a comparable training time for each iteration with that of LIBMF.

DAAL-MF-SGD has a better convergence speed than LIBMF, using less iterations to achieve the same convergence.
CPU utilization and Memory Bandwidth on KNL

• DAAL-MF-SGD utilizes more than 95% of all the 256 threads on KNL

• DAAL-MF-SGD uses more than half of the total bandwidth of MCDRAM on KNL

• We need to explore the full usage of all of MCDRAM’s bandwidth (around 400 GB) to further speed up DAAL-MF-SGD
Intra-node Performance: Haswell Xeon vs. KNL Xeon Phi

DAAL-MF-SGD has a better performance on KNL than on Haswell CPU, because it benefits from:

- KNL’s AVX512 vectorization
- High Memory Bandwidth

KNL has:
- \(3x\) speeds up by vectorization
- \(1.5x \text{ – } 4x\) speeds up to Haswell
Machine Learning using Harp Framework
The Growth of Model Sizes and Scales of Machine Learning Applications

2014

2015

- LDA - Topic Model
- MF - Matrix Factorization
- CNN - Convolutional Neural Network
- DNN - Deep Neural Network
*GPU cores
Challenges of Parallelization Machine Learning Applications

• Big training data
• Big model
• Iterative computation, both CPU-bound and memory-bound
• High frequencies of model synchronization
Types of Machine Learning Applications and Algorithms

- **Expectation-Maximization Type**
  - K-Means Clustering
  - Collapsed Variational Bayesian for topic modeling (e.g. LDA)

- **Gradient Optimization Type**
  - Stochastic Gradient Descent and Cyclic Coordinate Descent for classification (e.g. SVM and Logistic Regression), regression (e.g. LASSO), collaborative filtering (e.g. Matrix Factorization)

- **Markov Chain Monte Carlo Type**
  - Collapsed Gibbs Sampling for topic modeling (e.g. LDA)
Inter/Intra-node Computation Models
Model-Centric Synchronization Paradigms

- Synchronized algorithm
- The latest model

(A)

- Synchronized algorithm
- The latest model

(B)

- Asynchronous algorithm
- The stale model

(C)

- Asynchronous algorithm
- The stale model

(D)
Case Study: LDA mines topics in text collection

- Huge volume of Text Data
  - Information overloading
  - What on earth is inside the TEXT Data?
- Search
  - Find the documents relevant to my need (ad hoc query)
- Filtering
  - Fixed info needs and dynamic text data
- What's new inside?
  - Discover something I don't know
LDA and Topic model

- Topic Models is a modeling technique, modeling the data by probabilistic generative process.
- Latent Dirichlet Allocation (LDA) is one widely used topic model.
- Inference algorithm for LDA is an iterative algorithm using share global model data.

Document

Word

Topic: semantic unit inside the data

Topic Model
- documents are mixtures of topics, where a topic is a probability distribution over words

Normalized co-occurrence matrix

Mixture components

Mixture weights

SALSA
The Comparison of LDA CGS Model Convergence Speed

The parallelization strategy can highly affect the algorithm convergence and the system efficiency. This brings us four questions:

- **What part of the model needs to be synchronized?**
  The parallelization needs to decide which model parts need synchronization.

- **When should the model synchronization happen?**
  In the parallel execution timeline, the parallelization should choose the time point to perform model synchronization.

- **Where should the model synchronization occur?**
  The parallelization needs to tell the distribution of the model among parallel components, what parallel components are involved in the model synchronization.

- **How is the model synchronization performed?**
  The parallelization needs to explain the abstraction and the mechanism of the model synchronization.

**rtt & Petuum:** rotate model parameters  
**lgs & lgs-4s:** one or more rounds of model synchronization per iteration  
**Yahoo!LDA:** asynchronously fetch model parameters
**Inter-node Computation Models**

*(Training Data Items Are Partitioned to Each Process)*

### Computation Model A
- Once a process trains a data item, it locks the related model parameters and prevents other processes from accessing them. When the related model parameters are updated, the process unlocks the parameters. Thus the model parameters used in local computation are always the latest.

### Computation Model B
- Each process first takes a part of the shared model and performs training. Afterwards, the model is shifted between processes. Through model rotation, each model parameter is updated by one process at a time so that the model is consistent.

### Computation Model C
- Each process first fetches all the model parameters required by local computation. When the local computation is completed, modifications of the local model from all processes are gathered to update the model.

### Computation Model D
- Each process independently fetches related model parameters, performs local computation, and returns model modifications. Unlike A, workers are allowed to fetch or update the same model parameters in parallel. In contrast to B and C, there is no synchronization barrier.
Intra-node: Schedule Data Partitions to Threads
(only Data Partitions in Computation Model A, C, D; Data and/or Model Partitions in B)

(A) Dynamic Scheduler
- All computation models can use this scheduler.
- All the inputs are submitted to one queue.
- Threads dynamically fetch inputs from the queue.
- The main thread can retrieve the outputs from the output queue.

(B) Static Scheduler
- All computation models can use this scheduler.
- Each thread has its own input queue and output queue.
- Each thread can submit inputs to another thread.
- The main thread can retrieve outputs from each task’s output queue.
Harp Framework

Harp is an open-source project developed by Indiana University.

- MPI-like collective communication operations that are highly optimized for big data problems.
- Harp has efficient and innovative computation models for different machine learning problems.
## Data Types

### Arrays & Objects

#### Primitive Arrays
- ByteArray, ShortArray, IntArray, FloatArray, LongArray, DoubleArray

#### Serializable Objects
- Writable

### Partitions & Tables

#### Partition
- An array/object with partition ID

#### Table
- The container to organize partitions

#### Key-value Table
- Automatic partitioning based on keys
APIs

**Scheduler**
- DynamicScheduler
- StaticScheduler

**Collective**
- broadcast
- reduce
- allgather
- allreduce
- regroup
- pull
- push
- rotate

**Event Driven**
- getEvent
- waitEvent
- sendEvent
Case Study: LDA and Matrix-Factorization Based on SGD and CCD

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Node Type</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Xeon E5 2699 v3 (each uses 30 Threads)</td>
<td>Xeon E5 2670 v3 (each uses 20 Threads)</td>
<td></td>
</tr>
<tr>
<td>clueweb1</td>
<td>Harp CGS vs. Petuum (30)</td>
<td>Harp CGS vs. Petuum (60)</td>
<td></td>
</tr>
<tr>
<td>clueweb2</td>
<td>Harp SGD vs. NOMAD (30)</td>
<td>Harp SGD vs. NOMAD (60)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Harp CCD vs. CCD++ (30)</td>
<td>Harp CCD vs. CCD++ (60)</td>
<td></td>
</tr>
</tbody>
</table>
Collapsed Gibbs Sampling for Latent Dirichlet Allocation

**CGS Algorithm for LDA**

<table>
<thead>
<tr>
<th>Input:</th>
<th>training data $X$, the number of topics $K$, hyperparameters $\alpha, \beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>topic assignment matrix $Z$, topic-document matrix $M$, word-topic matrix $N$</td>
</tr>
</tbody>
</table>

1. Initialize $M, N$ to zeros
2. for document $j \in [1, D]$ do
3.     for token position $i$ in document $j$ do
4.         $Z_{ij} = k \sim Mult(\frac{1}{K})$
5.         $M_{kj} += 1; N_{wk} += 1$
6.     end for
7. end for
8. repeat
9.     for document $j \in [1, D]$ do
10.    for token position $i$ in document $j$ do
11.       $M_{kj} -= 1; N_{wk} -= 1$
12.       $Z_{ij} = k' \sim p(Z_{ij} = k|\text{rest})$ \{sample a new topic by Eq. (2) using SparseLDA [8]\}
13.       $M_{kj'} += 1; N_{wk'} += 1$
14.    end for
15. end for
16. until convergence
Matrix Factorization

SGD Algorithm for MF

**Input:** training matrix $V$, the number of features $K$, regularization parameter $\lambda$, learning rate $\epsilon$

**Output:** row related model matrix $W$ and column related model matrix $H$

1. Initialize $W, H$ to $\text{UniformReal}(0, 1/\sqrt{K})$
2. repeat
3. for random $V_{ij} \in V$ do
4. \{L2 regularization\}
5. $error = W_{i*}H_{*j} - V_{ij}$
6. $W_{i*} = W_{i*} - \epsilon(error \cdot H_{*j}^T + \lambda W_{i*})$
7. $H_{*j} = H_{*j} - \epsilon(error \cdot W_{i*}^T + \lambda H_{*j})$
8. end for
9. until convergence

CCD Algorithm for MF

**Input:** training matrix $V$, the number of features $K$, regularization parameter $\lambda$

**Output:** row related model matrix $W$ and column related model matrix $H$

1. Initialize $W, H$ to $\text{UniformReal}(0, 1/\sqrt{K})$
2. Initialize residual matrix $R$ to $V - WH$
3. repeat
4. for $V_{*j} \in V$ do
5. for $k = 1$ to $K$ do
6. $s^* = \frac{\sum_{i \in V_{*j}} (R_{ij} + H_{kj}W_{ik})W_{i*}}{\sum_{i \in V_{*j}} (\lambda + W_{i*}^2)}$
7. $R_{ij} = R_{ij} - (s^* - H_{kj})W_{i*}$
8. $H_{kj} = s^*$
9. end for
10. end for
11. for $V_{*i} \in V$ do
12. for $k = 1$ to $K$ do
13. $z^* = \frac{\sum_{j \in V_{i*}} (R_{ij} + W_{ik}H_{kj})H_{kj}}{\sum_{j \in V_{i*}} (\lambda + H_{kj}^2)}$
14. $R_{ij} = R_{ij} - (z^* - W_{ik})H_{kj}$
15. $W_{ik} = z^*$
16. end for
17. end for
18. until convergence
Features of Model Update in Machine Learning Algorithms

I. The algorithms can converge even when the consistency of a model is not guaranteed to some extent.

II. The update order of the model parameters is exchangeable.

III. The model parameters for update can be randomly selected.

Algorithm Examples

- Collapsed Gibbs Sampling for Latent Dirichlet Allocation
- Stochastic Gradient Descent for Matrix Factorization
- Cyclic Coordinate Descent for Matrix Factorization
A Parallelization Solution using Model Rotation

Maximizing the effectiveness of parallel model updates for algorithm convergence

Minimizing the overhead of communication for scaling

Input: $P$ workers, data $D$, model $A^0$, the number of iterations $T$

Output: $A^T$

1. parallel for worker $p \in [1, P]$ do
2. for $t = 1$ to $T$ do
3. for $i = 1$ to $P$ do
4. $A_{p}^{t_i} = F(D_p, A_{p}^{t_i-1})$
5. rotate $A_{p}^{t_i}$
6. end for
7. end for
8. end parallel for

Training Data $D$ on HDFS
Pipeline Model Rotation

Worker 0
- $A_{0a}$
- $A_{0b}$
- $A_{2a}$
- $A_{2b}$
- $A_{1a}$
- $A_{1b}$
- $A_{0a}$
- $A_{0b}$

Worker 1
- $A_{1a}$
- $A_{1b}$
- $A_{2a}$
- $A_{2b}$
- $A_{0a}$
- $A_{0b}$
- $A_{1a}$
- $A_{1b}$

Worker 2
- $A_{2a}$
- $A_{2b}$
- $A_{0a}$
- $A_{0b}$
- $A_{1a}$
- $A_{1b}$
- $A_{2a}$
- $A_{2b}$

Time

Shift
Dynamic Rotation Control for LDA CGS and MF SGD

Model Parameters From Rotation

Model Related Data

Multi-Thread Execution

Computes until the time arrives, then starts model rotation

Other Model Parameters From Caching
CGS Model Convergence Speed

<table>
<thead>
<tr>
<th>LDA Dataset</th>
<th>Documents</th>
<th>Words</th>
<th>Tokens</th>
<th>CGS Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>clueweb1</td>
<td>76163963</td>
<td>999933</td>
<td>299140784</td>
<td>$K = 10000$, $\alpha = 0.01$, $\beta = 0.01$</td>
</tr>
</tbody>
</table>

$K$: number of features; $a$, $b$ hyperparameters;
**SGD Model Convergence Speed**

<table>
<thead>
<tr>
<th>MF Dataset</th>
<th>Rows</th>
<th>Columns</th>
<th>Non-ZeroElements</th>
<th>SGD Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>clueweb2</td>
<td>76163963</td>
<td>999933</td>
<td>15997649665</td>
<td>$K = 2000, \lambda = 0.01, \epsilon = 0.001$</td>
</tr>
</tbody>
</table>

60 nodes x 20 threads/node

30 nodes x 30 threads/node

$K$: number of features; $\lambda$ regularization parameter; $\epsilon$ learning rate
**CCD Model Convergence Speed**

<table>
<thead>
<tr>
<th>MF Dataset</th>
<th>Rows</th>
<th>Columns</th>
<th>Non-Zero Elements</th>
<th>CCD Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>clueweb2</td>
<td>76163963</td>
<td>999933</td>
<td>15997649665</td>
<td>$K = 120, , \lambda = 0.1$</td>
</tr>
</tbody>
</table>

- 60 nodes x 20 threads/node
- 30 nodes x 30 threads/node

$K$: number of features; $\lambda$ regularization parameter
Outline

1. Motivation: Machine Learning Applications
2. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
3. Harp-DAAL Framework: Design and Implementations
4. Conclusions and Future Work
Harp-DAAL: High Performance Machine Learning Framework

**Harp**
1. Java API
2. Local computation: Java threads
3. Communication: Collective MapReduce

**DAAL**
1. Java & C++ API
2. Local computation: MKL, TBB
3. Communication: MPI & Hadoop & Spark

**Harp-DAAL**
1. Java API
2. Local Computation: DAAL
3. Communication: Collective MapReduce
Harp-DAAL in the HPC-BigData Stack

Harp-DAAL is at the intersection of HPC and Big Data stacks, which requires:

- **Interface**: User friendly, consistent with other Java written Data analytics Apps.
- **Low level Kernels**: highly optimized for HPC platforms such as many-core architecture
- **Models**: inherit Harp’s computation models for different ML algorithms
Inter-node test is done on two Haswell E5-2670 v3 2.3GHz nodes. We vary the size of input points and the number of centroids (clusters) By using DAAL-Harp’s high performance kernels, DAAL-Harp-Kmeans has a 2x to 4x speeds up over Harp-Kmeans
The Inter-node test is done on two Haswell E5-2670 v3 2.3GHz nodes. We use two datasets

- MovieLens, a small set with 9301274 points
- YahooMusic a large set with 252800275 points

For both datasets, we have around 5% to 15% speed up by using DAAL-SGD within Harp.

There are still some overheads of interfacing DAAL and Harp, which requires further investigation.
We decompose the training time into different phases. There are two overheads of interface:

- Harp-DAAL Interface
  - Conversion between data structures
- JNI interface
  - Data movement from Java heap to out-of-heap buffer for C++ native kernels in DAAL

The two overheads could take up to 25% of the total training time, which must be optimized in the future work.

- Rewrite some Harp codes to create shared memory space between DAAL and Harp
- Add more Harp compatible data structures to DAAL
Outline

1. Motivation: Machine Learning Applications
2. Harp-DAAL Framework: Design and Implementations
3. A Faster Machine Learning solution on Intel Xeon/Xeon Phi Architectures
4. Conclusions and Future Work
The Models of Contemporary Big Data Tools
Programming Models
Comparison of Iterative Computation Tools

Spark
- Implicit Data Partitioning
- Implicit Communication

Harp
- Explicit Data Partitioning
- Explicit Communication
  Various Collective Communication Operations

Parameter Server
- Explicit Data Partitioning
- Implicit Communication
  Asynchronous Communication Operations

Harp: a Hadoop plug-in based on map-collective models

- MPI-like collective communication operations that are highly optimized for big data problems.
- A Hadoop Plug-in to integrate with the ecosystems.
- Efficient and innovative computation models for different machine learning problems.
Hadoop/Harp-DAAL: Prototype and Production Code

Source codes available on Indiana University’s Github account

An example of MF-SGD is at
https://github.iu.edu/IU-Big-Data-Lab/DAAL-2017-MF-SGD

- Harp-DAAL follows the same standard of DAAL’s original codes
- improve DAAL’s existed algorithms for distributed usage
- add new algorithms to DAAL’s codebase.
- Harp-DAAL’s kernel is also compatible with other communication tools.
Summary and Future Work

• Identification of **Apache Big Data Software Stack** and integration with **High Performance Computing Stack** to give **HPC-ABDS**
  o ABDS (Many Big Data applications/algorithms need HPC for performance)
  o HPC (needs software model productivity/sustainability)

• Identification of **4 computation models** for machine learning applications

• **HPC-ABDS Plugin Harp**: adds HPC communication performance and rich data abstractions to Hadoop by development of Harp library of **Collectives** to use at Reduce phase
  o Broadcast and Gather needed by current applications
  o Discover other important ones (e.g. Allgather, Global-local sync, Rotation pipeline)

• Integration of **Hadoop/Harp** with Intel **DAAL** and other libraries

• Implement efficiently on each platform (e.g. Amazon, Azure, Big Red II, Haswell/KNL Clusters)