High-level Program Optimization for Data Analytics

Yufei Ding

North Carolina State University
Motivation: Faster Data Analytics

Source: oracle, 2012

Source: SAS, 2013
Role of My Research

Compiler Technology
- automatic, but mostly focus on instruction-level inefficiency in program implementations.

Automatic
High-Level
Program Optimization

(Big) Data Analytics + Other Data-intensive Applications
- high-level transformations, which is often more effective, but requires a huge amount of manual efforts.
My Research

High-level Program Optimization:
• Implementation → Algorithm; Instruction → Formula

Algorithmic Optimization for Distance-Related Problems
[ICML’15, VLDB’15, ICDE’17, PLDI’17]

Autotuning Algorithmic Choice for Input Sensitivity
[PLDI’15]

Generalizing Loop Redundancy Elimination at a Formula Level
[OOPSLA’17]

Examining Compilation Scheduling of JIT-Based Runtime System
[ASPLOS’14]

Parallel Stochastic Gradient Descent (SGD) with Sound Combiners
[applied for patent]
Focus of this talk

Automatic Algorithmic Optimization for Distance-Related Problems

maginudes of speedups.

VLDB’15, ICML’15, ICDE’2017, PLDI’17
Distance-related Problems

- These algorithms are widely used.

<table>
<thead>
<tr>
<th>Problems</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMeans</td>
<td>Data Mining</td>
</tr>
<tr>
<td>KNN (K Nearest Neighbor)</td>
<td>(Among Top 10 Most Popular DM Algorithms)</td>
</tr>
<tr>
<td>KNN Join</td>
<td>Image Processing</td>
</tr>
<tr>
<td>ICP (Iterative Search Problem)</td>
<td></td>
</tr>
<tr>
<td>P2P (Point-to-Point Shortest Path)</td>
<td>Graphics</td>
</tr>
<tr>
<td>Nbody</td>
<td>Computational Physics</td>
</tr>
</tbody>
</table>

- Distance computations are the performance bottleneck.
How to build a automatic framework to save all these manual efforts?
Challenges

• What are the beneficial and legal higher-level transformations that lead to better algorithms?
• Can we have an abstraction to unify various problems in different domains?
  → Then we should be able to turn the algorithmic optimization into compiler-based transformations.

Could have saved many years’ of manual efforts!
Case Study on KMeans

Yinyang KMeans:
A Drop-In Replacement of the Classic KMeans with Consistent Speedup

Collaborated w/ MSR
(Madan Musuvathi’s group)
Background: $K$Means

- Usage: group $N$ points (in $D$ dimensions) into $K$ clusters.
- Demo with $N = 600$, $D = 2$, $K = 4$
Background: KMeans

- **Usage:** group \( N \) points (in \( d \) dimensions) into \( K \) clusters.
- **Standard KMeans** [by Lloyd in 1957]:
  
  **I. Point Assignment:** Assign points to clusters based on \( d(p, c) \ \forall p, c \)
  
  **II. Center Update:** Update centers w/ new centroids.

  **Convergence**

  **Step I** is the performance bottleneck: \( N*K \) distances calc. per iteration.
Prior Works

• Grouping:
  - e.g., K-D Tree \cite{Kanungo2002}.
  - Overhead grows exponentially with dimension.

• Incremental Computing:
  - e.g., Triangle inequality \cite{Elkan2003, Hamerly2010, DrakeHamerly2012}.
  - Large Memory Overhead.
  - Slowdowns for medium dim, large K & N.

• Approximation \cite{Wang2012}
  - Unable to inherit the level of trust.

Standard KMeans by Lloyd remains the dominant choice in practice!
Yinyang KMeans

Grouping + Incremental Computing
• On average, **9.36X faster** than classic Means.
• **No slowdown** regardless of N, K, d.
• Guarantee to produce the **same result** as Standard KMeans.

Yin: upper bound
V.S.
Yang: lower bound
(thes bounds comprises the filters for distance computations)

A harmony w/ contrary forces
Triangular Inequality (TI)

- The fundamental tool for getting bounds:

\[ |d(q,c') - d(c',c)| \leq d(q,c) \leq d(q,c') + d(c',c) \]

**Lower bound:**
\[ \text{lb}(q, c) = |10 - 1| = 9 \]

**Upper bound:**
\[ \text{ub}(q, c) = 10 + 1 = 11 \]
How are bounds used?

Example: Will q switch its assignment from $c'_1$ to $c'_2$ in the next iteration? (q is currently assigned to $c_1$.)

Conclusion: No, because $ub(q, c'_1) < lb(q, c'_2)$.
Design of Yinyang Kmeans

- Innovative way of using upper and lower bounds.
  - Joint of filters: Group(Global) filter + Local filter.

Group (Global) filter + Local filter
Global Filtering

Filtering rule: if \( ub(q, c'_1) \leq lb(q, G') \), then \( q \) will not change its assignment.

How to compute these bounds?

\[
ub(q, c'_1) = ub(q, c_1) + \Delta c_1 \\
lb(q, G') = lb(q, G) - \max(\Delta c_i), \forall c_i
\]

Benefits: over 60\% redundant distance computation can be removed

Limiting factors of lower bound:
1. \( lb(q, G) \): closest center in \( G' \) (e.g., \( c_2 \)).
2. \( \max(\Delta c_i) \): biggest drifter (how far a center moved across iteration) (e.g., \( \Delta c_5 \)).

Current iter: \( G' = \{C'_1, C'_2, ..., C'_k\} - C'_1 \)

Next iter: \( G = \{C_1, C_2, ..., C_k\} - C_1 \)

(\( q \) is currently assigned to \( C_i \))
Group Filtering

Filtering Rule: if $ub(q,c') \leq lb(q, G'_i)$, then $q$ will not change its assignment to any center in $G'_i$.

How to compute these bounds?

$$ub(q,c_1) = ub(q, c'_1) + \Delta c_1$$

$$lb(q, G_i) \leq lb(q,G'_i) - \max (\Delta(c)), \forall c \in G_i$$

Benefits of $m$ lower bounds:
1. $lb(q, G_i)$: local closest center in $G_i$.
2. $\max (\Delta(c))$: local biggest drifter.

Over 80% redundant distance computations can be removed
Group Filtering

• Overhead Analysis (m groups):

\[
\begin{align*}
\text{ub}(q,c'_1) &= \text{ub}(q, c_1) + \Delta c_1 \\
\text{lb}(q, G'_i) &\leq \text{lb}(q, G_i) - \max (\Delta c), \forall c \in G_i
\end{align*}
\]

Time Cost: K distances (for center drifts) + N \cdot (m + 1) bounds lightweight compared to standard N \cdot K distances.

Space Cost: N \cdot (m + 1) for maintaining m lower bounds per point. comparable to N \cdot D for storing N points in D dimension.
Group Filtering

• When/How to group centers?

  one time grouping over initial centers through 5-iter Kmeans.

• How many groups?

  A space-conscious elastic design:
  \[ m = \begin{cases} 
  \frac{K}{10} & \text{if space allows} \\
  \text{max value} & \text{otherwise} 
  \end{cases} \]

Grouping centers into \( m \) groups: \( \{ G_1, G_2, \ldots, G_m \} \)
Filtering rule: for each center in the remaining group,
if \( \text{Min}(q, G'_i) \leq \text{lb}(q, G_i) - \Delta c_j \),
then \( q \) will not change its assignment to \( c_j \).

No extra memory cost!

Efficiency:
Over 90% redundant distance computations can be removed.
Evaluation

- Compared to three other methods:
  - Standard (Lloyd’s) K-Means
  - Elkan’s K-Means [2003]
  - Drake’s K-Means [2012]

- Input: real-world data sets (with different N, K, Dim)
  - 4 from UCI machine learning repository [Bache Lichman, 2013]
  - 4 other commonly used image data sets [Wang et al., 2012].

- Implemented in GraphLab (a map-reduce framework)
  - http://research.csc.ncsu.edu/nc-caps/yykmeans.tar.bz2

- Two machines
  - 16GB memory, 8-core i7-3770K processor
  - 4GB memory, 4-core Core2 CPU
Evaluation

Clustering results are the same as those of the standard Kmeans.

Baseline: Standard Kmeans
Evaluation

Baseline: Classic K-means (16GB, 8-core)
Towards Yinyang K-means on GPU
by Vadim Markovtsev 26 July 2016

The code: GitHub.

Yinyang K-Means: A Replacement
45 points by jcr 549 days ago | hide | past | web

Towards Yinyang K-means on GPU (sourced.tech)
61 points by tanoku 170 days ago | hide | past | web | 14 comments | favorite

Olivier Grisel @ogrisel
Yinyang K-Means: A Drop-In Replacement for the Classic K-Means faster than Stanza

datatonic @teamdatatonic
Yinyang K-Means: A Drop-In Replacement for the Classic K-Means with Consistent Speedup

Barney Pell @barneyp

Algorithmic Optimization Design

- **Triangle Inequality Optimization (TOP).**
  - **landmark definition.**
    - “*temporal landmarks*” for iterative problems like KMeans.
  - **group filtering.**
    - # of **groups** to strike a good tradeoff between space cost and redundant distance computation elimination.
TOP: Enabling Algorithmic Optimizations for Distance-related Problems

Collaborated w/ MSR (Madan Musuvathi’s group)

VLDB’2015
TOP Framework

- Compiler
  - problem semantic
  - TOP API
  - building blocks
    - Opt Lib

Abstract Distance-Related Problem

Variants of TI Optimizations

Our Analysis and Abstraction

- KNN
- KNN join
- ICP
- NBody
- KMeans
- Shortest Distance
• A 5-element Tuple <Q, T, D, R, C>
  • finding some kind of Relations between two sets of points, a Query set and a Target set, based on certain type of Distance and under some update Constraints.

<table>
<thead>
<tr>
<th>Problems</th>
<th>Query</th>
<th>Target</th>
<th>Distance</th>
<th>Relation</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>KMeans</td>
<td>Points</td>
<td>Centers</td>
<td>Euclidean</td>
<td>Top 1 Closest</td>
<td>Iterative update to T</td>
</tr>
</tbody>
</table>
KMeans Written with Our APIs

```
TOP_defDistance(Euclidean); // distance definition
T = init();
changedFlag = 1;
while (changedFlag){
    // find the closest target (a point in T) for each point in S
    N = TOP_findClosestTargets(1, S, T);
    TOP_update(T, &changedFlag, N, S); // T gets updated
}
```
TOP Framework

Abstract Distance-Related Problem

Our Analysis and Abstraction

Variants of TI Optimizations

Compiler

building blocks

TOP API

problem semantic

Opt Lib

KNN
KNN join
ICP
NBody
KMeans
Shortest Distance
Key Optimization Knobs

- Landmark definitions.
  - e.g., temporal landmark (e.g., Kmeans), spatial landmark (e.g., KNN).

- Number of groups/landmarks
- Order of Comparison

Beat the algorithms manually optimized by experts!
Optimization Selection

• 7 principles of applying TI optimization.

• Rule-based Selection Framework (Clang).

1. **Decide** the best way of defining landmark and the number of landmarks to use.
2. **Insert** codes preparing landmarks for optimizations.
3. **Replace** these TOP APIs (e.g., TOP_findClosestTargets) with optimized codes.
Evaluation

• Tested on six distance-related problems.

• Compared to two other methods for each problem:
  – Standard version without TI optimization.
  – Manual optimization from previous works.

• Input: real-world data sets used in previous papers.

• Machine
  – Intel i5-4570 CPU and 8G memory.
Evaluation — Running time

Each point in the graph stands for one input setting.

Baseline: Standard versions

Average speedups: 50X for TOP vs. 20X for manual version from previous works.

Over 93% of the distance computation can be saved by TOP.
• Theoretic foundation for generalization of TI to compute bounds of “vector dot product”!

\[ \vec{q} \cdot \vec{t} \geq |\vec{q}| \cdot |\vec{t}| \cdot \cos(\theta_{qL} + \theta_{tL}) \]

\[ \vec{q} \cdot \vec{t} \leq |\vec{q}| \cdot |\vec{t}| \cdot \cos(\theta_{qL} - \theta_{tL}) \]

- Deep learning, e.g., *Restricted Boltzmann Machines.*
  Text mining which uses cosine similarity as “distances”.
- Computation results is not directly used for comparison.

• Static analysis to detect code patterns for optimization (Clang).
A fundamental tension: Redundancy and Regularity.
  – critical for performance.

Our solution:
  – Careful implementations on GPU,
  – Elastic algorithmic design,
  – Up to 12X speedups over the state-of-art version (CUBLAS).
My Research

High-level Program Optimization:
- Implementation → **Algorithm**; Instruction → **Formula**

**Algorithmic Optimization for Distance-Related Problems**
[ICML’15, VLDB’15, ICDE’17, PLDI’17]

**Autotuning Algorithmic Choice for Input Sensitivity**
[PLDI’15]

**Generalizing Loop Redundancy Elimination at a Formula Level**
[OOPSLA’17]

**Examining Compilation Scheduling of JIT-Based Runtime System**
[ASPLOS’14]

**Parallel Stochastic Gradient Descent with Sound Combiners**
[applied for patent]
Other work

Autotuning Algorithmic Choice for Input Sensitivity

Collaborated w/ MIT
(Saman Amarasinghe's group)
Algorithmic Autotuning

- Best optimization: autotuning + alg. choices.
  - E.g., what is best optimization for sorting?

  - Huge number of potential optimizations by varying the type and order of algorithm to use.
Our Contribution

• 3X averaged speedup over static optimization
  – on 6 benchmarks (e.g., sorting, clustering, helmholtz).

• *Language and compiler support.*

• A *Two-level input learning framework*
  – the enormous optimization space,
  – variable accuracy of algorithmic choices.
My Research

High-level Program Optimization:
• Implementation \(\rightarrow\) Algorithm; Instruction \(\rightarrow\) Formula

Algorithmic Optimization for Distance-Related Problems
[ICML’15, VLDB’15, ICDE’17, PLDI’17]

Autotuning Algorithmic Choice for Input Sensitivity
[PLDI’15]

Generalizing Loop Redundancy Elimination at a Formula Level
[OOPSLA’17]

Examining Compilation Scheduling of JIT-Based Runtime System
[ASPLOS’14]

Parallel Stochastic Gradient Descent with Sound Combiners
[applied for patent]
Future Research (Long-Term, High-Level)

**Applications**

**Platform**

**Metric**

- Security
- Resilience
- Energy
- Accuracy
- Performance (running time)
- Multicore
- GPU
- IoT
- Cloud

**Automatic Higher-level (algorithmic) Optimization**

- Data mining
- Graph analysis
- Deep learning
- Computational Physics
- Computational Economics
Future Research (3~5 Years)

– Combine the higher-level program optimization and lower-level optimizations. *(High-performance Computing)*

– Automate extractions of the domain-specific knowledge for high-level program optimizations. *(NLP, text mining, ontology, ...)*

– Combine algorithmic optimizations with approximation-based computing?

– Deep learning in bioinformatics, astronomy, etc.
  - Hyper-parameter tuning + structural learning
  - Incremental computing for the searching process.

– Cyber-Physical Systems *(CPS)*.
  - Program language support for expressing user specifications, e.g., helping resolve dependency in smart homes.
  - Embedded intelligence: data analytics in edge computing *(IoT)*.
Publications


