Rethinking Compilers in the Rise of Machine Learning and AI

Xipeng Shen

Computer Science, North Carolina State University
The journey of a snowflake

Born of a raindrop, laced by the breeze, a snowflake forms. Spinning through space, dancing through trees; “I’m not water anymore; I’m a beautiful snowflake” Proudly, it announces its new identity, while falling into a lake, melt. “Welcome home, child”, gently says the water.
Traditional Optimizing Compilers

A C compiler understands C constructs; hardcoded basic knowledge (e.g. linear algebraic identities); hardcoded optimization heuristics.

Derived program properties
- def-use;
- dependence;
- control flow;
- ...

Low-level code transformations
- loop transformations
- common subexpression elimination
- vectorization/parallelization
- ...

A C program.
Thesis of this Talk

• High-level semantic is key for unleashing the hidden power of compilers
• AI is key for unleashing the hidden power of high-level semantic-driven compilations
High-Level Semantic

• Beyond the semantic of the primitive constructs in a general purpose language
• Importance has been well perceived
  • Numerous Domain Specific Languages (DSL)
• Potential for generalizing compiler techniques
  • *Generalized strength reduction*
  • *Generalized loop redundancy removal*
Generalized Strength Reduction

[PLDI’17, ICDM’17, ICDE’17, VLDB’15, ICML’15]

w/ Madan Musuvathi & Todd Mytkowicz
Strength Reduction

\[ \frac{b}{2} \rightarrow b \gg 1 \]

Traditional: only instruction level.
Triangular Inequality:
\[ a - b \leq d \leq a + b \]
Reduce expensive distance calculations to bounds calculations.

9 \leq d \leq 11

Distances: O(K) V.S. O(N * K)
KMeans

ICML’2015

NIPS’2012

SIAM’2010

ICML’2003

Using the Triangle Inequality to Accelerate $\ell$-Means

Charles Elkan
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92037

Abstract

The $\ell$-means algorithm is by far the most widely used method for discovering clusters in data. We show how to accelerate it dramatically, while still preserving exactly the same result as the standard algorithm. The acceleration algorithm avoids unnecessary distance calculations by exploiting the triangle inequality in two different ways, and by keeping track of lower upper bounds for distances between points and centers. Experiments show that the new algorithm is significantly faster than the standard algorithm.

...
Our Generalization

• Conceptual connections between TI & strength reduction
• Extend TI theory
  – Angle triangle inequality (ATI)
  – Its relations with traditional TI based on edge (ETI)
• Create *Generalized Strength Reduction* as a compiler technique
  – For distance & dot product
  – Offer systematic deployment of TI
  – Enable automatic *algorithm optimizations*
• Speedups: tens to hundreds of times
ATI

- Vector dot product

\[ \vec{q} \cdot \vec{t} = |\vec{q}| \cdot |\vec{t}| \cos \theta_{qt} \]

\[ \theta_{qL} + \theta_{tL} \geq \theta_{qt} \geq \theta_{qL} - \theta_{tL} \]

**Angle Tri. Ineq. (ATI)**

Holds in spaces of any (>1) dimensions!
ATI always gives tighter distance bounds than TI does!

(detailed proof in PLDI’17 paper).
Usage of ATI

• Example: Restricted Boltzmann Machine (RBM).
Usage of ATI

Hidden Layer: \( h \)

Visible Layer: \( v \)

- **visible -> hidden**
  - Conditional probability.
    \[
    P(h_m = 1) = \sigma(\vec{v} \cdot W(:,m))
    \]
    \( \sigma(t) \) sigmoid function.
  - Set \( h_m \).
    \[
    h_m = \begin{cases} 
    1 & \text{if } r < P(h_m = 1) \\
    0 & \text{otherwise}
    \end{cases}
    \]
    \( r \): a random number \([0,1]\).
Usage of ATI

• When and how bounds can be used as a replacement?

\[
h_m = \begin{cases} 
1 & \text{if } r < P(h_m = 1) \\ 
0 & \text{otherwise} 
\end{cases}
\]

If \( lb(P(h_m = 1)) > r \), then \( h_m = 1 \),

else if \( ub(P(h_m = 1)) < r \), then \( h_m = 0 \).

In both cases, bounds can help avoid computing \( P(h_m = 1) \).
Complexities of Applying ETI + ATI

• ATI V.S. ETI
  – Tightness is not the only factor relevant to the benefits of TI-

Solution: Guided TI Adaptation to choose the best TI optimization on the fly

(details in PLDI’17 & VLDB’15).
Overall Speedup (No quality loss)

- Over default implementation.
Over Manual TI Opt

K-Means (K=1024)

Baseline: Classic K-means
(16GB, 8-core Intel Ivy Bridge)

Yinyang K-Means

Code link in ICML'15 paper.

Datasets (size, dim)
Example II: Generalized Loop Redundancy Removal

[OOPSLA’17]
Loop Redundancy

\[
\text{for } (i=0; i<N; i++)\
\{\
a[i] = b/c;\
\}\n\]
Motivating Example

Any Redundant Computations?

```c
w = w0;
while (d > 0.01) {
    d = 0;
    for (i = 0; i < M; i++) {
        d += a[i] + b[i] * w;
    }
    w = w - 0.001 * d;
}
```

Stay the same across `while` loop, but vary across `for` loop.

Stay the same across `for` loop, but change across `while` loop.

Elusive to existing techniques.
• An equivalent form of the example.

Motivating Example

\[
\begin{align*}
\text{w} &= \text{w}_0; \\
\text{while } (d > 0.01) \{ \\
& \quad \text{A} = \sum_i a[i]; \\
& \quad \text{B} = \sum_i b[i]; \\
& \quad d = A + B \cdot w; \\
& \quad w = w - 0.001 \cdot d \\
\}
\end{align*}
\]

\[
\begin{align*}
\text{w} &= \text{w}_0; \\
\text{while } (d > 0.01) \{ // \ K \ iterations \\
& \quad d = 0; \\
& \quad \text{for } (i = 0; i < M; i++) \{ \\
& \quad \quad d += a[i] + b[i] \cdot w; \\
& \quad \} \\
& \quad w = w - 0.001 \cdot d; \\
\}
\end{align*}
\]

\[
\begin{align*}
\mathcal{O}(M\cdot K) & \rightarrow \mathcal{O}(M + K)
\end{align*}
\]
Key Observation

Nature of such hidden redundancies:

Their discovery and removal require large-scoped analysis and computation reordering at both expression and loop levels.
Generalized Loop Redundancy Elimination (GLORE)

A Set of Novel Algorithms
- Operand folding
- Alternative form generation
- Closure-based algorithm
- Minimum-union algorithm
- Loop encapsulation

LER Notation
- Formulae
  - ...
  - ...
  - ...

Enabler

Transformer

Optimized Loop
- Opt Formulae
  - ...
  - ...
  - ...
Optimization through GLORE algorithms

LER notation

\[
\mathbf{w}_t \sum_{i=1}^{N} (a[i] + b[i] \cdot w_t) \Rightarrow d_t.
\]

Code-to-formula Transformation

Formula-to-Code Transformation

\[
\sum_{i=1}^{N} a[i] \Rightarrow \text{tmp1},
\]

\[
\sum_{i=1}^{N} b[i] \Rightarrow \text{tmp2},
\]

\[
\mathbf{w}_t (\text{tmp1} + \text{tmp2} \cdot w_t) \Rightarrow d_t.
\]
Reasons for the Algorithm Designs

• Variety of loop redundancies
• Complex math operations (e.g., div, sin, mod)
• Loop index dependence
• Many possible orders of computations
  • Interplays between reordering of expressions and that of loops

In general, finding the best order is NP-complete. [Chi-chung+:1997]
Entire Work Flow

See OOPLSA’17 for details.
Evaluation: 21 Benchmarks

Real-world problems

Batch gradient descent (BGD) a nonlinear partial differential equation (PDE).

Stencil benchmarks

Dibligbilharm, Diso3X3, Imorph, Noise1, Iyoki, Inevatia, and Dresid.

Programs from prior work


Pluto benchmark suite

ssymm, fuse, priv2, fmri, and ccsd.
Categories 3 & 4 (Redundant Loops)

Reduced Computational Complexity

No prior methods work on them.

1.15-1.8X speedups on other programs.
Reflection

Right abstractions plus suitable algorithms may go a long way in leveraging high-level semantics and hence advancing compiler technology.

Can the idea be generalized to a broader range of high-level semantics?
Thesis of this Talk

- High-level semantic is key for unleashing the hidden power of compilers
- AI is key for unleashing the hidden power of high-level semantic-driven compilations
What other high-level semantics?

• A lot...
  • Each library API defines some high-level semantics

• How are current compilers treating them?
  • Black boxes
  • Barriers for program analysis and optimizations
object AScalaSparkJob {
    def main(args: Array[String]) {
        val conf = new SparkConf()
        conf.setAppName("My First Spark Scala Application")
        val ctx = new SparkContext(conf)
        val file = "file://home/Crimes_-Aug-2015.csv";
        val logData = ctx.textFile(file, 2)
        val numLines = logData.filter(line => true).count()
        val vectors = logData.map(…)
        val kMeansModel = KMeans.train(vectors, 2, 20)
    }
}
Lots of Prior Efforts

- Interprocedural analysis
  - Limited by analyzability
- Specification-based solutions
  - Leveraging high-level semantic conveyed by manual specifications
- Telescoping Languages (Kennedy et al.), Broadway (Guyer & Lin), Speckle (Vandevoorde et al.), ...
Telescoping Languages

• On programs in S languages
  • 10—143X speedups
  • On one program, 23 000X using structure of a matrix

[IEEE 2005]
“If each domain library effectively defines a new language, then there will be an enormous number of specialized high-level languages that will each require an optimizing compiler.”

—Kennedy et al. [IEEE 2005]
Telescoping Languages

A generator of domain-specific optimizers.

Library Specifications

Transformation specifications

Library Analysis and Preparation Phase

Bottleneck for adoption.

Script Compilation Phase

(Adapted from Kennedy+:IEEE’2005)
Key Barrier: Knowledge

- The optimizer generator needs to know
  - Knowledge on the APIs
    - Functional, behavioral, performance attributes of each API
    - Relations (e.g., algebraic equivalence) of sequences of APIs
  - Knowledge on the transformations
    - Precondition, effects, and cost
- Difficulties
  - Difficult to write or maintain
  - Hard to be correct or complete
Existing Efforts

• Manual efforts
• Automatic derivations of code specifications
  • R. Bodik et al., T. Xie et al., others
    • Code, documentations, dynamic observations.
    • State machines, Graphs, NLP, ML, etc.
    • Focused on invariants or code contract
• Much more needed
  • (Algebraic) relations between APIs, equivalence of sequences, context-sensitive performance properties, data dependences, code transformations, ...
Other Open Challenges

• How to best transform the library and program
  • Explosion of the set of abstract types and specialized versions
  • Best ways and order to apply transformations
Revisit in Lens of Modern AI & ML

• Automatic discover high-level semantics
  • knowledge discovery from multi-sources
    • Documentations of APIs & comments
    • Code, dynamic observations
• Automatic discover & apply high-level transformations
  • Learning from experience
  • Planning, reasoning, searching
Telescoping Languages

A program in scripting language

Software building blocks with high-level semantic understandable to compilers.

HW → TelGen → An efficient implementation upon the library
Automatic Programming

High-level user intention, intuitively expressed.

An efficient implementation of an efficient algorithm.

Software building blocks with high-level semantic understandable to autocoder.
Holy Grail of Optimizing Compilers

• A full freedom to explore
  • Algorithm
  • Implementation
• Not bounded by
  • Hardcoded algorithm opaque to compilers
  • Aliases and other code complexities
  • Abstraction walls
Benefits to Users

- Programming productivity
- Less human involvements: Less error prone
  - Reliability
  - Security
  - Maintainability
- Analyzability-Preserving Compositions
A long-time goal

“Automatic programming has been a goal of computer science and artificial intelligence since the first programmer came face to face with the difficulties of programming.”

Myth 1: End-user-oriented automatic programming systems do not need domain knowledge.

“End-user-oriented automatic programming systems must be domain experts. There is no point at which someone who knows nothing about programming communicates directly with someone who knows nothing about the application domain.”
Myth 2: End-user-oriented, general purpose, fully automatic programming is possible.

“…such an automatic programming system would have to be expert in every application domain. Unfortunately, artificial intelligence is nowhere near supporting this superhuman level of performance.”

Are there more hope given the new progress of AI and ML? Can those domain knowledge be automatically learned?
Three Key Questions to Answer

• What does user see?
• What does autocoder know?
• How does autocoder work?

All should be revisited in the lens of modern AI and ML.
An Example

“Natural languages are an attractive choice for communication between end users and an automatic programming system.

... Unfortunately, enabling machines to converse in natural language is way beyond the current capabilities of artificial intelligence.”

[MIT AI Memo]
v39: “The New Compiler”
v349, v353, v379,
v443, v453, ...
Lambda papers.
Summary

**Abstraction & Generalization**

Traditional compiler/Low-level method
Analysis and optimizations based on primitive language constructs.

**High-level semantic-driven compilation**
Analysis and optimizations at a larger scope and a higher level.

**AI, ML, etc.**

Generators of high-level semantic based compilers/Autocoder
Scalable approach to leveraging high-level semantics.
Final Message

• Being open-minded, embracing new directions
• Try new perspectives on compiler constructions
• Try to pursue systematic approaches to high-level semantic-driven compilations
  • Rather than being satisfied with another DSL