Towards Intelligent Programming Systems for Modern Computing

Xipeng Shen

Computer Science, North Carolina State University
Unprecedented Scale

Y2011
20 petaflops
10 mw power

Y202X
1000 petaflops
20 mw power

2020
35 zettabytes

as much Data and Content
Over Coming Decade

80%
Of world’s data is unstructured

50X perf
2X power!
Heterogeneity becomes Norm

Massively parallel accelerators are becoming ubiquitous.
Thesis

To address the challenges in modern computing, one of the keys exists in making programming systems more intelligent.

For advancing programming systems, right problem formulating goes a long way.
Modern Computing

**Application**
Data analytics, Machine learning, …

**Infrastructure**
Data centers, Cloud, IoT, …

**Architecture**
Heterogeneous parallel processors, Emerging complex memory, …

**TOP**
Algorithmic optimizer for data analytics
[VLDB’15, ICML’15]

**GStreamline+**
Memory optimization for GPU
[ASPLOS’11, Micro’14, ICS’16]
TOP:
Enabling Algorithmic Optimizations for Distance-Related Problems

Up to 100s X speedups.

VLDB’2015, ICML’2015
Role of Compiler

Learning Problem

Can compilers optimize algorithms?
Why algorithm level?

Reason 1:
Large benefits: orders of magnitude speedups at no extra cost.

Reason 2:
Compiler may outsmart ML experts.
**Example**

**Triangular Inequality:**

\[ a - b \leq d \leq a + b \]

---

**K-Means**
K-Means

Using the Triangle Inequality to Accelerate k-Means

Charles Elkan
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093-0114

ELKAN@CS.UCSD.EDU

Abstract
The $k$-means algorithm is by far the most widely used method for discovering clusters in data. We show how to accelerate it dramatically, while still always computing exactly the same result as the standard algorithm. The accelerated algorithm avoids unnecessary distance calculations by applying the triangle inequality in two different ways, and by keeping track of lower and upper bounds for distances between points and centers. Experiments show that the new algorithm is this center. Conversely, if a point is much closer to one center than to any other, calculating exact distances is not necessary to know that the point should be assigned to the first center. We show below how to make these intuitions concrete.

We want the accelerated $k$-means algorithm to be usable wherever the standard algorithm is used. Therefore, we need the accelerated algorithm to satisfy three properties. First, it should be able to start with any initial centers, so that all existing initialization methods can continue to be used. Second, given the same initial centers, it should al-
A Fast Exact $k$-Nearest Neighbors Algorithm for High Dimensional Search Using $k$-Means Clustering and Triangle Inequality

Xueyi Wang

Abstract—The $k$-nearest neighbors ($k$-NN) algorithm is a widely used machine learning method that finds nearest neighbors of a test object in a feature space. We present a new exact $k$-NN algorithm called $k$MANN ($k$-Means for $k$-Nearest Neighbors) that uses the $k$-means clustering and the triangle inequality to accelerate the searching for nearest neighbors in a high dimensional space. The $k$MANN algorithm has two stages. In the buildup stage, instead of using complex tree structures such as metric trees, $kd$-trees, or ball-tree, $k$MANN uses a simple $k$-means clustering method to preprocess the training dataset. In the searching stage, given a query object, $k$MANN finds nearest training objects starting from the nearest cluster to the query object and uses the triangle inequality to reduce the distance calculations. Experiments show that the performance of $k$MANN is surprisingly good compared to the traditional $k$-NN algorithm and tree-based $k$-NN algorithms such as $kd$-trees and ball-trees. On a collection of 20 datasets with up to $10^6$ records and $10^4$ dimensions, $k$MANN shows a 2- to 80-fold reduction of distance calculations and a 2- to 60-fold speedup over the traditional $k$-NN algorithm for 16 datasets. Furthermore, $k$MANN performs significantly better than a $kd$-tree based $k$-NN algorithm for all datasets and performs better than a ball-tree based $k$-NN algorithm for most datasets. The results show that $k$MANN is effective for searching nearest neighbors in high dimensional spaces.

[11] have been proposed to efficiently reduce the distance calculations and find exact nearest neighbors in higher dimensions. These methods iteratively divide training objects and build tree structures using criteria such as absolute coordinates and relative distances, so that a query object needs to check distances with only a limited number of training objects instead of the whole dataset. One problem for these methods is that when the dimensionality of a dataset is high, most of the training objects in the data structures will end up being evaluated and the searching efficiency is no better to or even worse than the traditional $k$-NN algorithm [10] [18], especially for large $k$ values.

Due to the difficulty of accelerating the $k$-NN algorithm in high dimensional space, some methods have focused on finding approximate answers. For example, the hashing method from [9] and the priority queue based method from [3] achieved a speedup of several fold over the traditional $k$-NN by outputting $k$ neighbors within $(1+\epsilon)$ of the true nearest neighbor distances. Hart [13] and Wilson [23] used techniques called condensing and editing to reduce objects from the dataset and accelerate the searching for nearest neighbors.
P2P: Point-to-Point Shortest Path

Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks

Computing the Shortest Path: A* Search Meets Graph Theory

Andrew V. Goldberg*  Chris Harrelson†

Abstract
We propose shortest path algorithms that use A* search in combination with a new graph-theoretic lower-bounding technique based on landmarks and the triangle inequality. Our algorithms compute optimal shortest paths and work on any directed graph. We give experimental results showing that the most efficient of our new algorithms outperforms previous algorithms, in particular A* search with Euclidean bounds, by a wide margin on road networks and on some synthetic problem families.

1 Introduction
The shortest path problem is a fundamental problem with numerous applications. In this paper we study one of the most common variants of the problem, where the goal is to find a point-to-point shortest path in a weighted, directed graph. We refer to this problem as the P2P problem. We assume that for the processing space. The best bound in this context (see [10]) is superlinear in the output path size unless the path is very long. Preprocessing using geometric information and hierarchical decomposition is discussed in [19, 28, 34]. Other related work includes algorithms for the single-source shortest path problem, such as [1, 4, 6, 7, 13, 14, 16, 17, 18, 22, 25, 32, 35], and algorithms for approximate shortest paths that use preprocessing [3, 23, 33].

Usually one can solve the P2P problem while searching only a small portion of the graph; the algorithm’s running time then depends only on the number of visited vertices. This motivates an output-sensitive complexity measure that we adopt. We measure algorithm performance as a function of the number of vertices on the output path. Note that this measure has the additional benefit of being machine-independent.

In Artificial Intelligence settings, one often needs to adapt pathfinding algorithms. The breadth-first

SIAO’05  ALENEX’04
Observations

• TI has led to many enhanced algorithms across problems and domains.

• Applying TI well is tricky, hence the many manual efforts and publications.

Thoughts

• Can we have an abstraction to represent all the problems?
• Can we then generalize the TI optimizations into compiler-based transformations?
Abstract Distance Problem
(Q, T, D, R, C)
Abstract Distance-Related Problem

Our Analysis and Abstraction

Essence & 7 Principles of TI Optimizations

KNN
KNN join
ICP
NBody
KMeans
Shortest Distance
Key Insights

• Reuse through Landmarks
• Spatial & temporal reuses
• Elasticity through hierarchical landmarks
• Efficient bounds update through ghosts for iterative alg.
• Order of comparison

See VLDB’15 for details.
**Usage**

- **TOP API**
- **Basic algorithm description**
- **Compiler**
- **Staged program code**
- **TI Opt Lib**
- **Efficient execution**

**TOP_defDistance**(Euclidean);
T = init();
changedFlag = 1;
while (changedFlag){
    N = **TOP_findClosestTargets**(1, S, T);
    **TOP_update**(T, &changedFlag, N, S);
}

**Systematic**

**Ad hoc**
K-Means (K=1024)

Clustering results are same as original method’s.

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

Yinyang K-Means
Code link in ICML’15 paper.
On K-Means

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

UKbench Speedup

- Elkan's
- Drake's
- TOP

Speedup (X)

Yinyang K-Means
On All Benchmarks

**Speedups**

- Manual Version
  - Speedups: $\times 0$, $\times 1$, $\times 10$, $\times 2$, $\times 10$, $\times 4$

- TOP Version
  - Speedups: $\times 10$, $\times 6$, $\times 10$, $\times 13$

**# distance calculations**

- Manual Version
  - Calculations: $10^2$, $10^3$, $10^4$

- TOP Version
  - Calculations: $10^2$, $10^3$, $10^4$

**Insight:**
The right abstraction and formulation turn a compiler into an automatic algorithm optimizer, giving out large speedups.

Average speedups: $50\times$ vs $20\times$.

Save at least 93% calculations.

*Intel i5-4570 CPU and 8G memory*
Modern Computing

Application
Data analytics, Machine learning, …

Infrastructure
Data centers, Cloud, IoT, …

Architecture
Heterogeneous parallel processors, Emerging complex memory, …

TOP
Algorithmic optimizer for data analytics
[VLDB’15, ICML’15]

GStreamline+
PORPLE
Memory optimization for GPU
[ASPLOS’11, Micro’14, ICS’16]
Overcome GPU Limitations

Zheng Zhang  
(Prof. @ Rutgers Univ)

Bo Wu  
(Prof. @ Colorado Mines)

Guoyang Chen  
(Qualcomm)
Graphic Processing Unit (GPU)

- Massive parallelism
- Favorable
  - computing power
  - cost effectiveness
  - energy efficiency
Challenges

- Scheduling Limitations
- Irregular Mem & Control
- Dyn Task Parallelism
Our Explorations

Compiler-based software solutions
Solutions

Scheduling Limitations

Irregular Mem & Control

Dyn Task Parallelism

SM-Centric & EffiSha [ics15,ppopp17]

GStreamline & PORPLE [asplos11,micro14, ics16]

FreeLaunch [micro15]

Compiler-based software solutions

Monday PPoPP Session 1
Dynamic Irregularities

Degrade throughput by up to \((\text{warp size} - 1)\) times.

\((\text{warp size} = 32 \text{ in modern GPUs})\)
Solution 1: Thread-Data Remapping

4 trans/warp

Irregularity in a warp: problematic; across warps: okey!

1 trans/warp

Principle of solution: Turn intra-warp irreg. into reg. or inter-warp irreg.
Trans-1: Data Reordering

\[ P[\ ] = \{0,5,2,3,2,3,7,6\} \]

\[ A[\ ]:\]

\[ Q[\ ] = \{0,1,2,3,2,3,6,7\} \]

\[ A'[\ ]:\]

**maintain mapping between threads & data values**

**tid**: thread ID;  
**\(\downarrow\)**: a thread;  
**\(\downarrow\)**: data access;  
**\(\downarrow\)**: data relocation

**original**

\[ ... = A[P[tid]]; \]

**<redirection>**

**transformed**

\[ ... = A'[Q[tid]]; \]

**<relocation>**

**<redirection>**
Trans-2: Job Swapping

- Job = operations + data elements accessed

**original**

\[ \text{newtid} = Q[tid]; \]
\[ P[ ] = \{0,5,2,3,2,3,7,6\} \]

**transformed**

\[ \text{newtid} = Q[tid]; \]
\[ \ldots = A[P[\text{newtid}]]; \]
\[ Q[ ] = \{0,4,2,3,1,5,6,7\} \]
G-Streamline  
[ASPLOS’2011 ]

First framework enabling runtime thread-data remapping.

CPU-GPU pipeline to hide transformation overhead.

Kernel splitting to resolve dependences.

1.08—2.5X speedups
Solution 2: Data Placement

Global memory
Texture memory
Shared memory
Constant memory
L1/L2 cache
Read-only cache
Texture cache
GPU Memory

- Global memory: coalescing; cache hierarchy
- Texture memory: 2D/3D locality; texture cache; read-only
- Shared memory: on-chip; bank conflicts
- Constant memory: broadcasting; cached; read-only
- L1/L2 cache: private/shared
- Read-only cache: read-only data
- Texture cache: 2D/3D locality; read-only
Data Placement Problem

- Global memory
- Texture memory
- Shared memory
- Constant memory
  - (L1/L2 cache)
  - (Read-only cache)
  - (Texture cache)

Data in a program

3X performance difference
Data Placement Problem

Properties:

- Machine dependent
  Changes across models/generations
- Input dependent
  Changes across runs

Options:

- Manual efforts by programmers?
- Offline autotuning?
PORPLE in a Whole

**OFFLINE**

- architect/user
- microkernels
- org. program

**MSL**
(mem. spec. lang.)

**PORPLE-C**
(compiler)

**access patterns**

**staged program**

**PLACER**
(placing engine)

**online profile**

**EFFICIENT EXECUTION**

**ONLINE**

- mem spec
- desired placement

More details in our Micro’2014 paper.
Properties of PORPLE

• Good portability to new memory
  • Just need new MSL spec
  • Program adapts automatically

• Adaptivity to new program inputs
  • On-the-fly placement with placement-agnostic code.

• Generality to regular & irregular programs
  • Static analysis + lightweight online profiling
GPU Models

- K20c
- M2075
- C1060
Potential for Future Memory Systems

3D Stacked Memory

Persistent Memory

DRAM (NUMA)
Final Takeaways

- **Large potential** of compilers for modern computing
- **Right problem formulation** is a key

**TOP**

An *algorithmic optimizer*. Up to 100x speedups.

**GStreamline**

**PORPLE**

Portable solution to mem. complexity. Consistent speedups cross GPUs.

**TOP Framework**

- Compiler
- TI Opt Library

**PORPLE**

architect

org. program

EFFICIENT EXECUTION

desired placement

MSL (mem. spec. lang.)

PORPLE-C (placering engine)

PLACER (placering engine)