Northwestern

Evaluation of K-Means Data Clustering Algorithm on Intel Xeon Phi

Sunwoo Lee, Wei-keng Liao, Ankit Agrawal, Nikos Hardavellas, and Alok Choudhary
EECS Department
Northwestern University

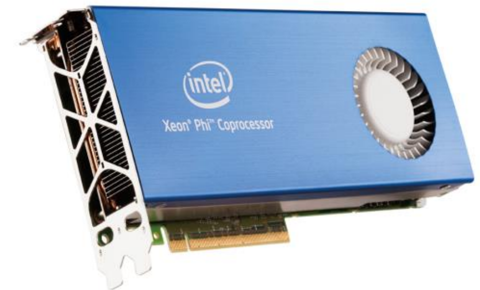
Contents

- Introduction
- Parallel K-Means Algorithm
- Techniques for Utilizing Xeon Phi’s Features
- Evaluation and Conclusion

Introduction
Intel Xeon Phi

- Intel Xeon Phi Processor
  - Many Integrated Core (MIC) architecture
  - Relatively weak computing cores (1~1.3GHz)
  - 57~72 physical cores with 4-way hardware threading

- Vectorization Capability
  - Instruction-level parallelism
    - ✓ 512-bit wide Vector Processing Units (VPUs)



Introduction

K-Means Data Clustering Algorithm

- Clustering Algorithm for Spatial Data
  - Partitions a dataset into k distinct groups
  - Each member data belongs to the cluster with the nearest mean

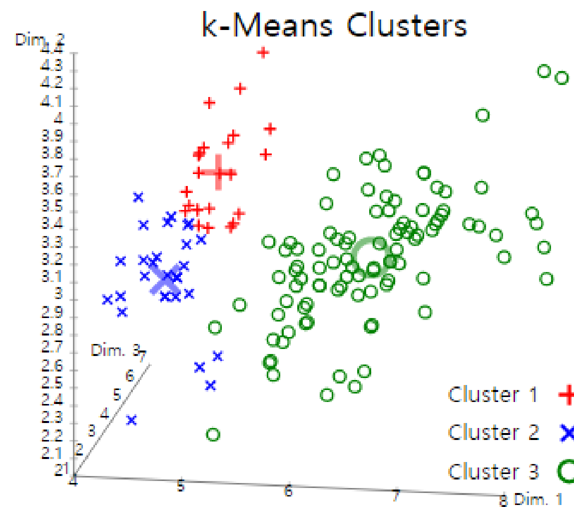
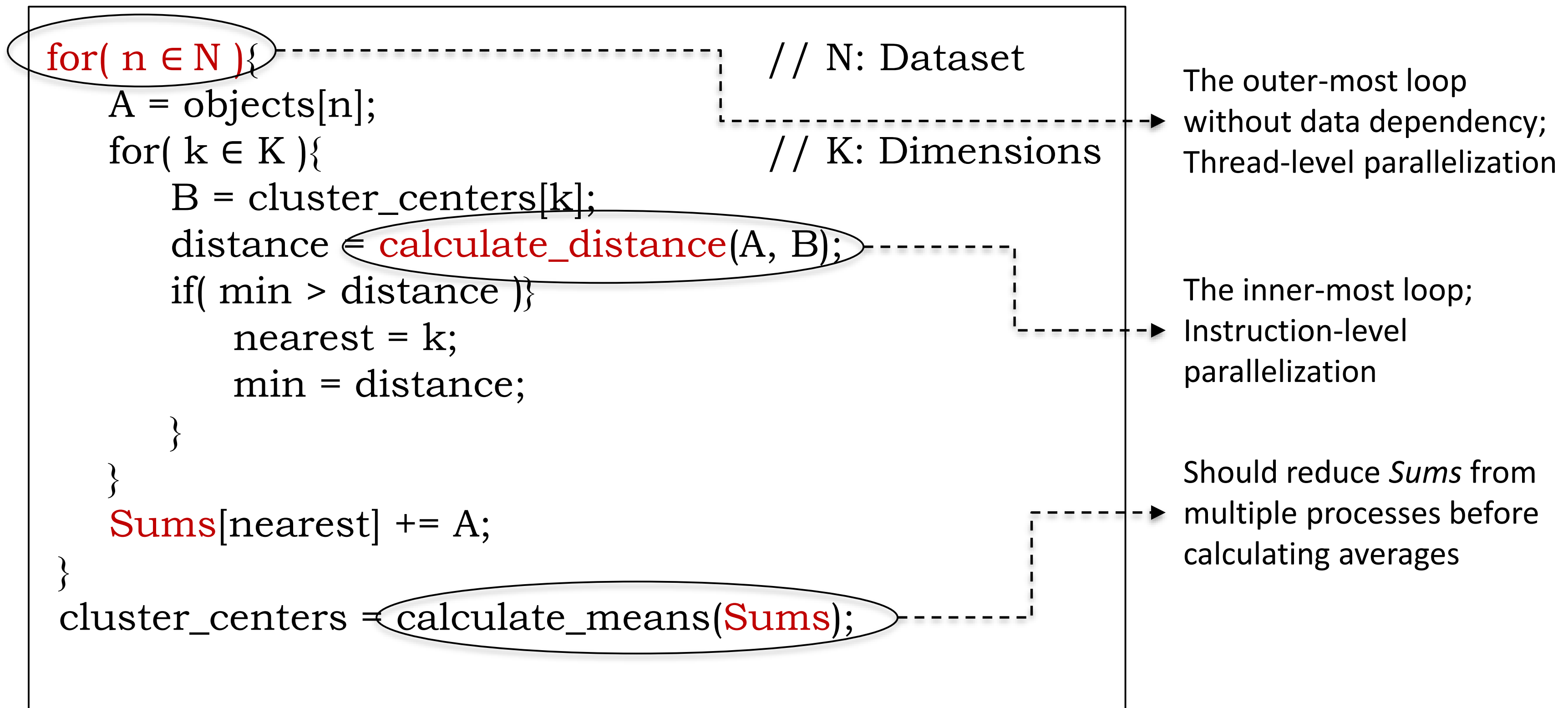


Figure Reference: https://en.wikipedia.org/wiki/K-means_clustering#/media/File:Iris_Flowers_Clustering_kMeans.svg

Parallel K-Means (1/2)

- Loop-based Structure



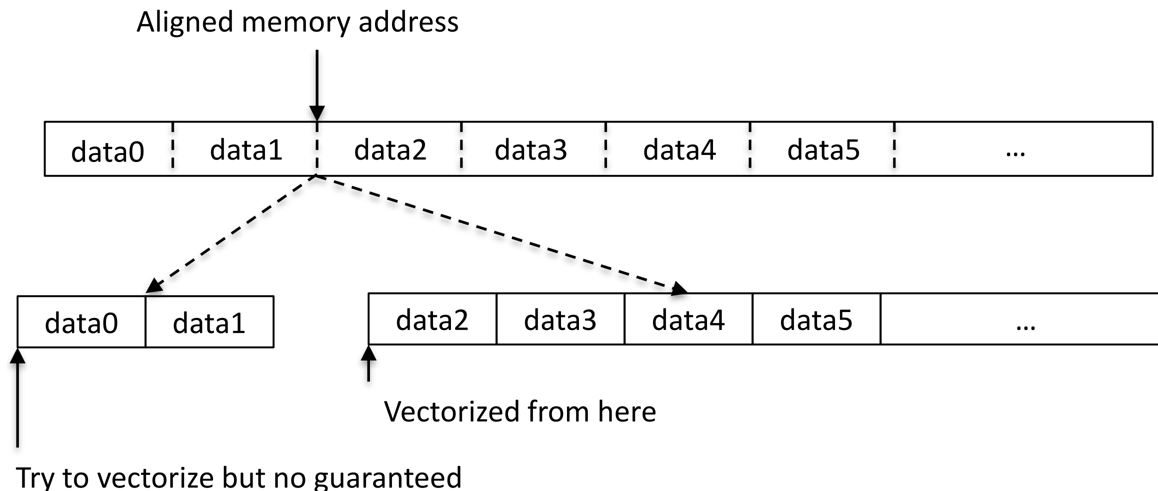
Parallel K-Means (2/2)

- Multi-level Parallelization Strategy
  - Distributed Memory
    - ✓ MPI
    - ✓ Distribute dataset to multiple machines
  - Shared Memory
    - ✓ OpenMP
    - ✓ Assign a subset of the given data to each thread
  - Instruction-level Parallelism
    - ✓ Single Instruction Multiple Data (SIMD) operations
    - ✓ Auto-vectorization of Intel compiler

Problems

- Naïve implementation of parallel K-Means engenders;
  - Low vectorization intensity
  - Low cache hit ratio

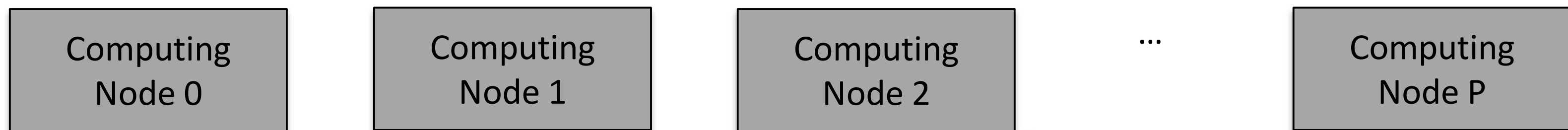
- Loop peelings



Efficient Vectorization

Memory Layout

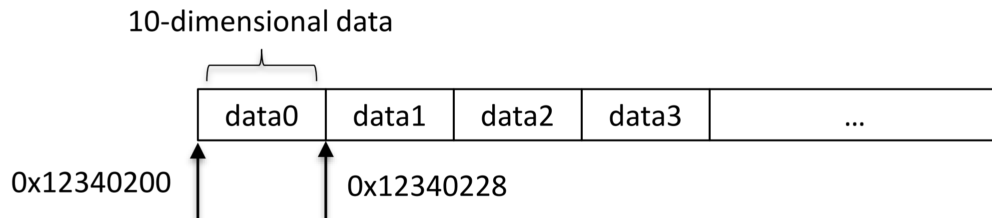
- 512-bit aligned contiguous memory spaces
  - Intel AVX-based VPUs require a memory alignment for optimal data movement for vector instructions



Efficient Vectorization

Data Padding (1/2)

- What if dimension is not divisible by VPU width?
  - E.g., distance calculation with 10-dimensional data points



Unaligned!
Loop peeling from the second iteration!

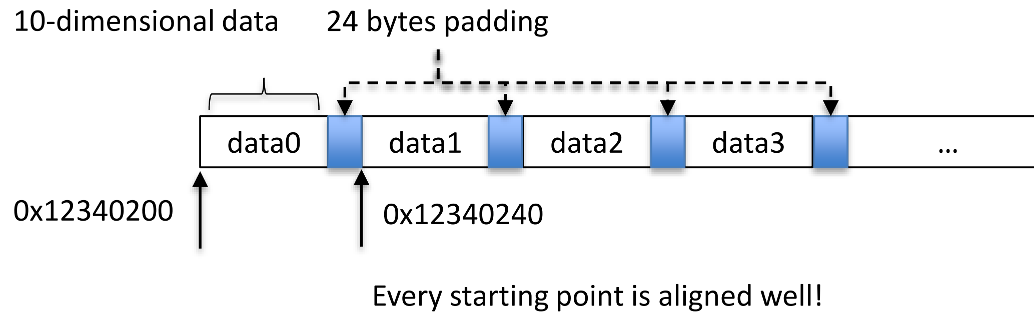
Algorithm 2 Efficient distance calculation on Xeon Phi with hints to compiler

1: float **objects is the input data points
2: float **clusters is the cluster centers
3: **for each** $i \in n$ **do**
4:     float sum = 0
5:     float *data_point = (float *) objects[i]
6:     **for each** $j \in k$ **do**
7:         float *cluster_center = (float *) clusters[i][j]
8:         __assume_aligned(data_point, 64)
9:         __assume_aligned(cluster_center, 64)
10:         **for** $c = 1, ..., numCoords$ **do**
11:             sum += (data_point[c] - cluster_center[c])^2

Efficient Vectorization

Data Padding (2/2)

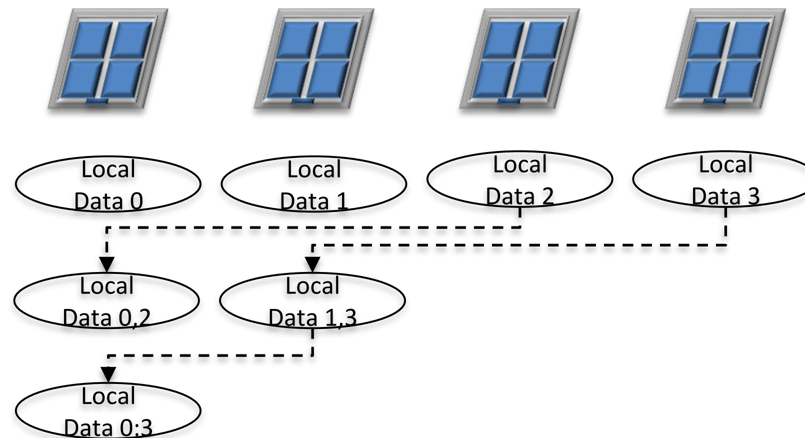
- Pad up each data point so as to make them aligned



- Every distance calculation is vectorized perfectly at the cost of more memory consumption

Cache-aware Reduction

- Averaging Coordinates for Calculating New Centroids
  - When adding two local data, one should be always its own local data → make data stays in cache as long as possible
  - Read the remote data from threads with close thread IDs → consecutive 4 threads share a L2 cache



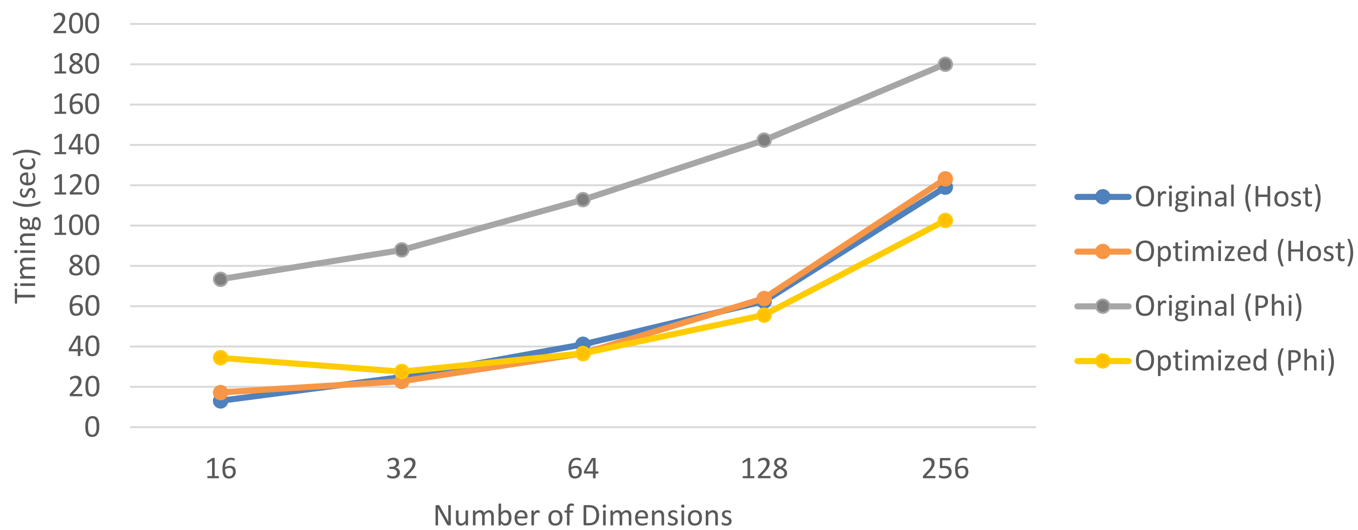
Experiment Setup

- Host System
  - Dual socket Intel Xeon processor, E5-2695 v3
  - 14 physical cores and 2-way SMP per core
  - 2296 MHz frequency of each core
- Xeon Phi
  - MIC architecture, Xeon Phi 7120 (Knight Corner)
  - 61 physical cores and 4-way SMP per core
  - 1238 MHz frequency of each core
- Dataset
  - Synthetic datasets with various dimensions (aligned/unaligned)
  - 4 millions /16 millions of data points with various dimensions
  - Data generator http://cucis.eecs.northwestern.edu/projects/DMS/MineBenchDownload.html

Evaluation

Performance Study with Aligned Dataset

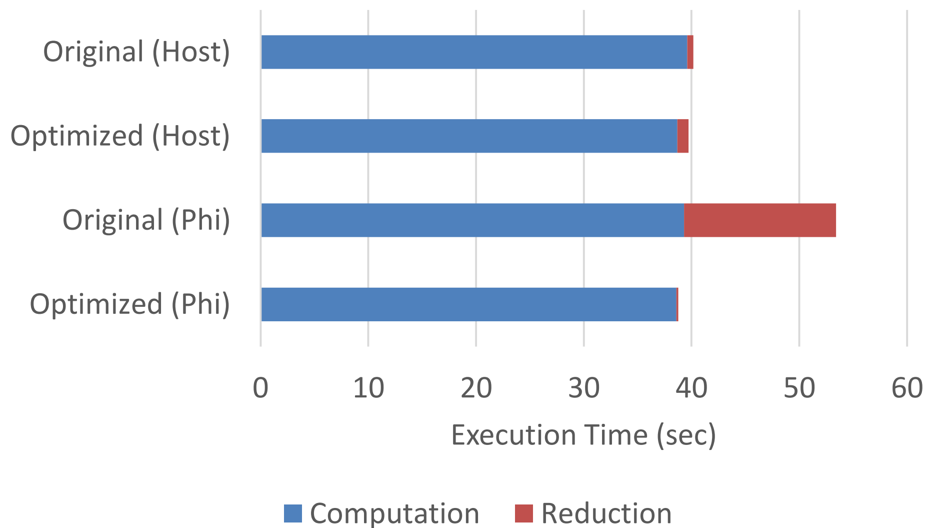
Dimension	16	32	64	128	256
Original (Phi)	73.37	87.9	112.79	142.3	179.99
Optimized (Phi)	34.41	27.56	36.55	55.58	102.5
Original (Host)	13.09	24.87	41.07	62.43	118.99
Optimized (Host)	17.18	22.79	36.55	63.89	123.08



Evaluation

Effect of Cache-aware Parallel Reduction

Works	Computation	Reduction
Original (Host)	39.62	0.55
Optimized (Host)	38.71	1.03
Original (Phi)	39.31	14.11
Optimized (Phi)	38.60	0.18



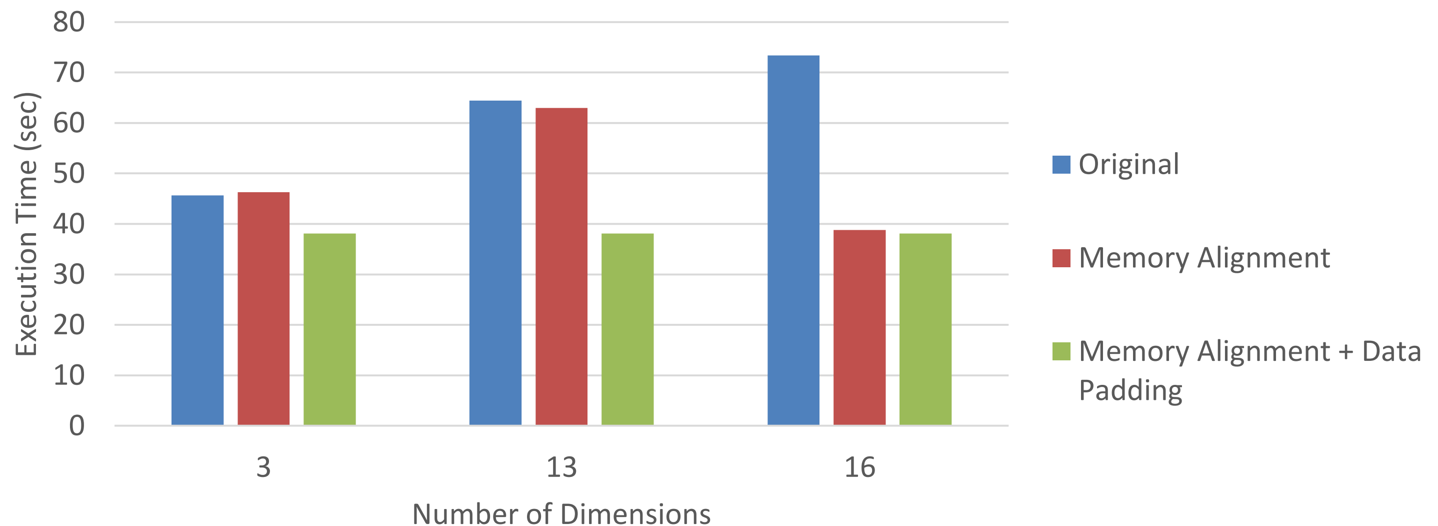
Performance Analysis with Intel VTune Amplifier Tools

Metric	Original (Phi)	Optimized (Phi)
Average CPI per Thread	3.51	2.82
Average CPI per Core	0.88	0.71
Vectorization Intensity	12.33	14.71
L1 Data Access Ratio	31.11	36.24
L2 Data Access Ratio	358.99	15628.93
L1 Hit Ratio	0.83	0.83
L1 TLB Miss Ratio	0.0011	0.0021

Evaluation

Performance Study with Unaligned Dataset

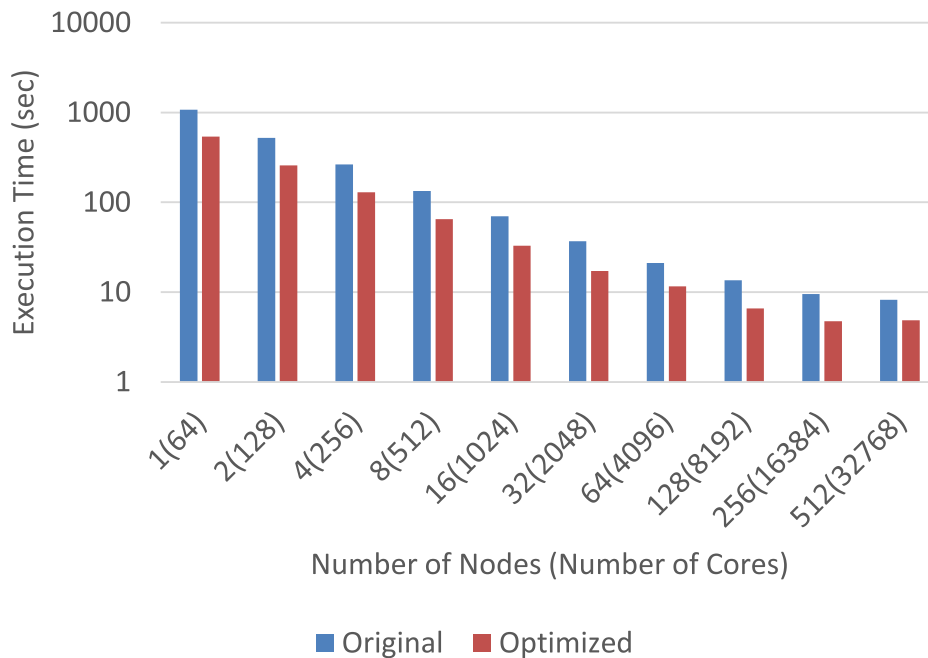
Dimensions	3	13	16
Original	45.65	64.41	73.37
Optimized 1	46.27	62.96	38.82
Optimized 2	38.12	38.11	38.07



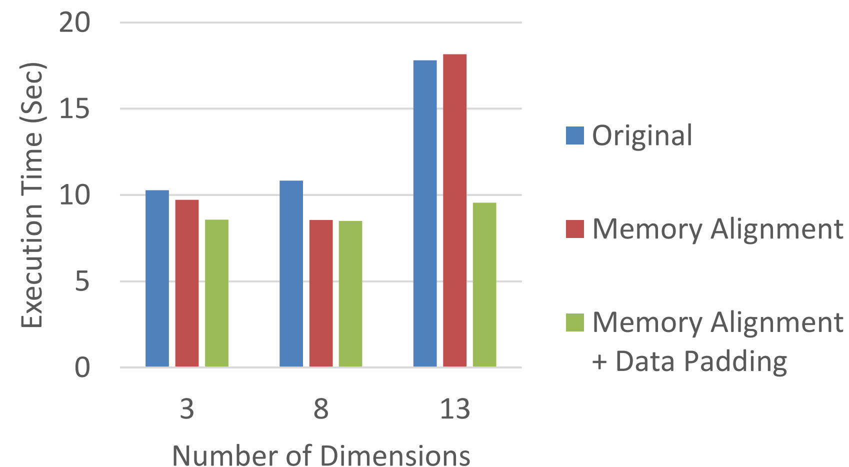
Evaluation

Performance Study on Multi-Node Supercomputer

Nodes	1	2	4	8	16	32	64	128	256	512
Original	1074.98	522.45	264.73	133.56	69.83	37.01	21.12	13.55	9.52	8.19
Optimized	539.64	258.82	129.31	65.13	32.92	17.23	11.64	6.59	4.73	4.87



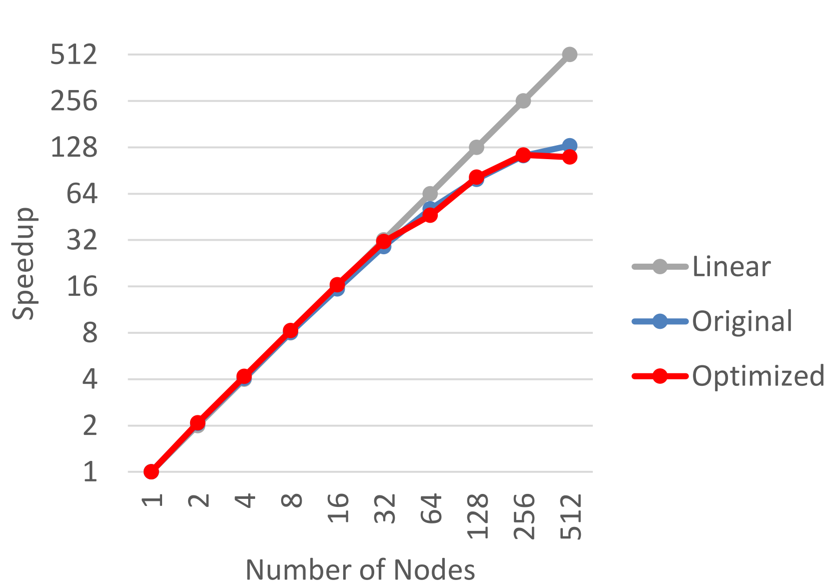
Dimensions	3	8	13
Original	10.28	10.83	17.81
Optimized 1	9.73	8.55	18.17
Optimized 2	8.58	8.49	9.55

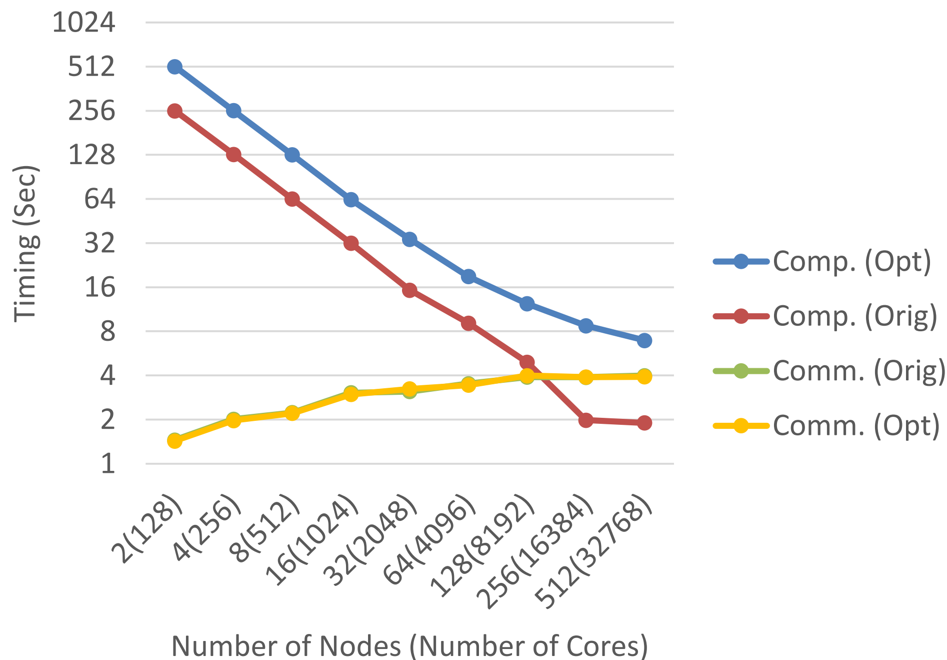


Evaluation

Multi Node Performance Study

Nodes	1	2	4	8	16	32	64	128	256	512
Original	1	2.06	4.06	8.05	15.39	29.04	50.91	79.36	112.86	131.25
Optimized	1	2.08	4.17	8.29	16.39	31.31	46.35	81.83	114.13	110.74





Conclusion

- To achieve a good performance on Intel Xeon Phi,
  - Efficient vectorization with an appropriate memory layout
  - Loop peelings can be avoided by padding each data point
    - Auto-vectorization can work better only with some hints
  - Efficient cache utilization in reduction
- Not only K-Means but also…
  - Any data mining algorithms with an iterative structure or sequential memory access pattern

Q&A