Large-Scale Automatic K-Means Clustering for Heterogeneous Many-Core Supercomputer

Teng Yu, Wenlai Zhao, Pan Liu, Vladimir Janjic, Xiaohan Yan, Shicai Wang Haohuan Fu, Guangwen Yang, John Thomson

汇报人：胡畅

College of Computer Science and Electronic Engineering

October 5, 2022

Outline

Outline

K-Means definition

K-Means Definition

Formalized, given n samples, $\mathcal{X}^d = \{x_i^d | x_i^d \in \mathcal{R}^d, i \in \{1, ..., n\}\}$, where $x_i^d = (x_{i1}, x_{i2}, ..., x_{id})$, We aim to find k d-dimensional centroids $\mathcal{C}^d = \{c_j^d | c_j^d \in \mathcal{R}^i\}, j \in \{1, ..., k\}$ to minimize the object $\mathcal{O}(\mathcal{C})$:

$$\mathcal{O}(\mathcal{C}) = \frac{1}{n} \sum_{i=1}^{n} dis(x_i^d, c_{a_i}^d)$$

$$a_i = argmin_{j \in \{1,..,k\}} dis(x_i^d, c_{a_i}^d)$$

$$1.: \ a(i) = arg\ min_{j \in \{1...k\}}\ dis(x_i^d, c_j^d)\ (Assign)$$

$$2.: \ c_j^d = \frac{\sum_{arg\ a(i)=j} x_i^d}{|arg\ a(i) = j|} \quad (Update)$$

Figure: KMeans algorithm steps

KMeans

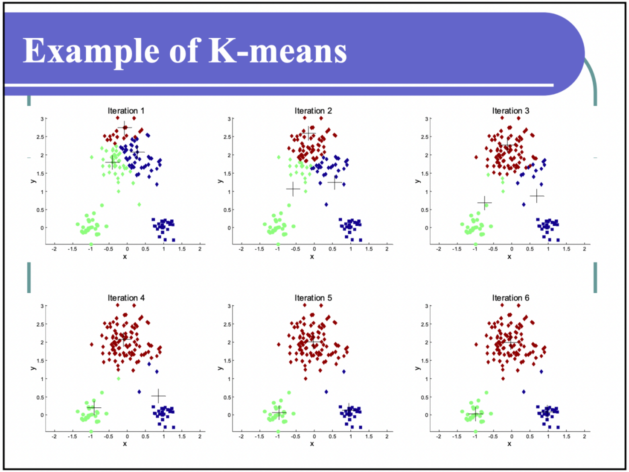

Figure: KMeans exmaple

Why parallel K-Means?

Finding the optimal solution for a general k-means problem is known to be NP-hard:

1. number of centroids (k)
2. number of dimensions (d)
3. proper hyper-parameters, such as the targeted number of centroids (k)

Introduction

- large-scale clustering problems with up to 196,608 dimensions and over 160,000 targeting centroids

Introduction

- large-scale clustering problems with up to 196,608 dimensions and over 160,000 targeting centroids
- automatic hyper-parameter determination

Introduction

- large-scale clustering problems with up to 196,608 dimensions and over 160,000 targeting centroids
- automatic hyper-parameter determination
- achieve high performance and scalability for problems

Introduction

- large-scale clustering problems with up to 196,608 dimensions and over 160,000 targeting centroids
- automatic hyper-parameter determination
- achieve high performance and scalability for problems
- support clustering without sufcient prior knowledge

Outline

Sunway TaihuLight

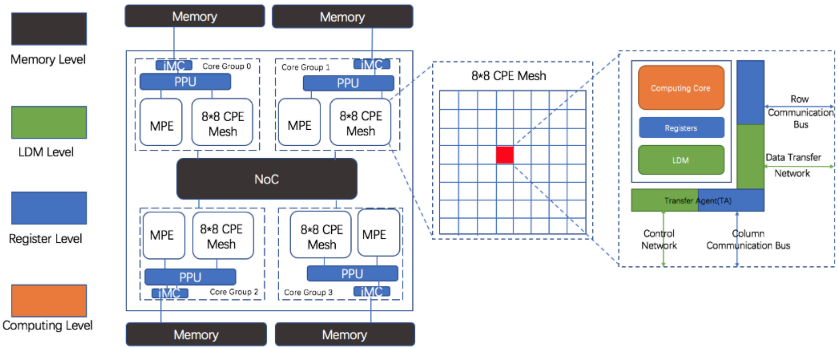

Figure: The general architecture of the SW26010 many-core processor

Three level parallelism

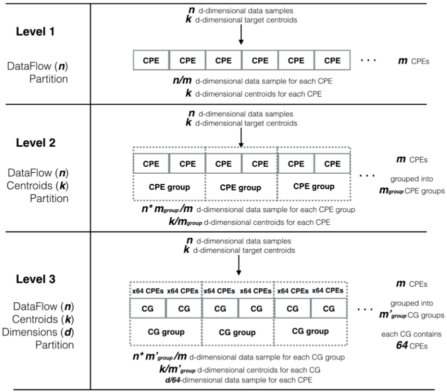

Fig. 2. Three-level k-$means$ design for data partition and parallelism on Sunway architecture

Figure: Three-level k-means design for data partition and parallelism on Sunway architecture

DataFlow Partition

3.1 Level 1 - DataFlow Partition

Algorithm 1 Basic Parallel *k-means*

INPUT: Input dataset $\mathcal{X} = \{x_i | x_i \in R^d, i \in [1, n]\}$, and initial centroid set $\mathcal{C} = \{c_j | c_j \in R^d, j \in [1, k]\}$
$P_l \xleftarrow{load} \mathcal{C},\ l \in \{1 \dots m\}$
repeat
  // Parallel execution on all CPEs:
  for $l = 1$ *to* m **do**
    Init a local centroids set $\mathcal{C}^l = \{c_j^l | c_j^l = \mathbf{0}, j \in [1, k]\}$
    Init a local counter $count^l = \{count_j^l | count_j^l = 0, j \in [1, k]\}$
    for $i = (1 + (l - 1) * \frac{n}{m})$ *to* $(l * \frac{n}{m})$ **do**
      $P_l \xleftarrow{load} x_i$
      $a(i) = arg\ min_{j \in \{1 \dots k\}} dis(x_i, c_j)$
      $c_{a(i)}^l = c_{a(i)}^l + x_i$
      $count_{a(i)}^l = count_{a(i)}^l + 1$
    end for
    for $j = 1$ *to* k **do**
      AllReduce c_j^l and $count_j^l$
      $c_j^l = \frac{c_j^l}{count_j^l}$
    end for
  end for
until $\mathcal{C}^l == \mathcal{C}$
OUTPUT: $\mathcal{C}$

Figure: Level 1 - DataFlow Partition

DataFlow Partition

```c
#include <omp.h>
#include <stdio.h>

#define len 10000
int data[len];
int threads[10];

int main() {

  for(int i = 0; i < len; i++) {
    data[i] = 1;
  }

  #pragma omp parallel for num_threads(10)
  for(int i = 0; i < len; i++) {
    printf("%d i = %d\n", (int) omp_get_thread_num(), i);
    threads[(int) omp_get_thread_num()] += data[i];
  }
  int sum = 0;
  for(int i = 0; i < 10; i++) {
    printf("threads[%d] = %d\n", i, threads[i]);
    sum += threads[i];
  }
  printf("sum = %d\n", sum);
  return 0;
}
```

Figure: Openmp code for parallel

Algorithm 2 Parallel k-$means$ for k-scale

INPUT: Input dataset $\mathcal{X} = \{x_i | x_i \in R^d, i \in [1, n]\}$, and initial centroid set $\mathcal{C} = \{c_j | c_j \in R^d, j \in [1, k]\}$

$P_l \xleftarrow{load} c_j \quad j \in (1 + mod(\frac{l-1}{m_{group}}) * \frac{k}{m_{group}}, (mod(\frac{l-1}{m_{group}}) + 1) * \frac{k}{m_{group}})$

repeat

  // Parallel execution on each CPE group $\{P\}_{l'}$:

  for $l' = 1$ to $\frac{m}{m_{group}}$ **do**

    Init a local centroids set $\mathcal{C}^{l'}$ and counter $count^{l'}$

    for i = $(1 + (l' - 1)\frac{n * m_{group}}{m})$ to $(l' \frac{n * m_{group}}{m})$ **do**

      $\{P\}_{l'} \xleftarrow{load} x_i$

      $a(i)' = arg\ min_j\ dis(x_i, c_j)$

      $a(i) = min.\ a(i)'$

      $c^{l'}_{a(i)} = c^{l'}_{a(i)} + x_i$

      $count^{l'}_{a(i)} = count^{l'}_{a(i)} + 1$

    end for

    for j = $(1 + mod(\frac{l-1}{m_{group}}) * \frac{k}{m_{group}})$ to $((mod(\frac{l-1}{m_{group}}) + 1) * \frac{k}{m_{group}})$ **do**

      AllReduce $c^{l'}_j$ and $count^{l'}_j$

      $c^{l'}_j = \frac{c^{l'}_j}{count^{l'}_j}$

    end for

  end for

until $\cup\, \mathcal{C}^{l'} == \mathcal{C}$

OUTPUT: $\mathcal{C}$

Figure: Level 2 - DataFlow and Centroids Partition

Algorithm 3 Parallel k-$means$ for k-scale and d-scale

INPUT: Input dataset $\mathcal{X} = \{x_i | x_i \in R^d, i \in [1, n]\}$, and initial centroid set $\mathcal{C} = \{c_j | c_j \in R^d, j \in [1, k]\}$

$CG_{l''} \xleftarrow{load} c_j^d,\ l'' \in \{1 \ldots \frac{m}{64}\}, j \in (1 + mod(\frac{l''-1}{m'_{group}}) * \frac{k}{m'_{group}},\ (mod(\frac{l''-1}{m'_{group}}) + 1) * \frac{k}{m'_{group}})$

repeat

  // Parallel execution on each CG group $\{CG\}_{l''}$:

  for $l'' = 1\ to\ \frac{m}{64}$ **do**

    Init a local centroids set $\mathcal{C}^{l''}$ and counter $count^{l''}$

    for i = $(1 + (l'' - 1)\frac{n * m'_{group}}{m})\ to\ (l''\frac{n * m'_{group}}{m})$ **do**

      for u = $(1 + mod(\frac{l-1}{64}) * \frac{d}{64}\ to\ (mod(\frac{l-1}{64}) + 1) * \frac{d}{64})$ **do**

        $CG_{l''} \leftarrow x_i\ (P_l \leftarrow x_i^u)$

      end for

      $a(i)' = arg\ min_j\ dis(x_i, c_j)$

      $a(i) = min.\ a(i)'$

      $c_{a(i)}^{l''} = c_{a(i)}^{l''} + x_i$

      $count_{a(i)}^{l''} = count_{a(i)}^{l''} + 1$

    end for

    for j = $(1 + mod(\frac{l''-1}{m'_{group}}) * \frac{k}{m'_{group}})\ to\ ((mod(\frac{l''-1}{m'_{group}}) + 1) * \frac{k}{m'_{group}})$ **do**

      AllReduce $c_j^{l''}$ and $count_j^{l''}$

      $c_j^{l''} = \frac{c_j^{l''}}{count_j^{l''}}$

    end for

  end for

until $\cup\, \mathcal{C}^{l''} == \mathcal{C}$

OUTPUT: $\mathcal{C}$

Figure: Level 3 - DataFlow and Centroids and Dimensions Partition

Determinig the optimal

$$r(k) = \inf\{t : \exists y_1, \ldots, y_k \text{ in } R^d, \mathcal{X}^d \subseteq \bigcup_{1 \le s \le k} B(y_s, t)\},$$

$$\Delta r'(k) = r'(k) - r'(k+1),$$

Figure: Formula of determinig the optimal k

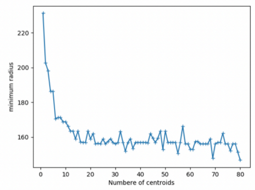

Figure: The evaluation function r'(k) to determine the optimal k value.

Outline

Experiment Results

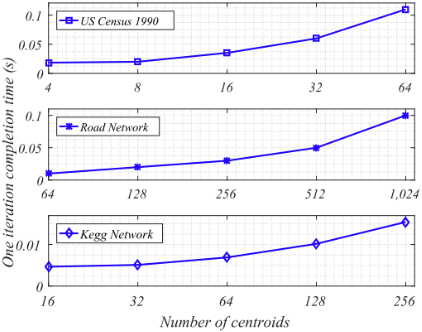

Figure: Level 1 - dataflow partition

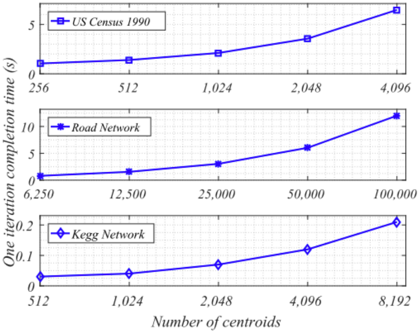

Figure: Level 2 - dataflow and centroids partition

Experiment Results

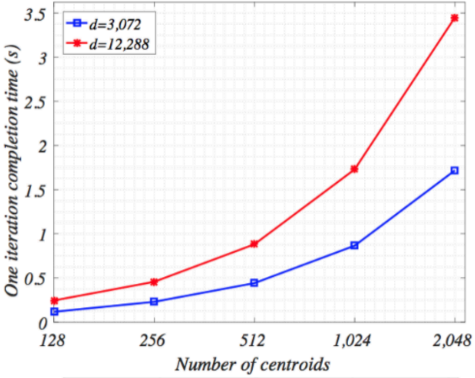

Figure: Level 3 - dataflow, centroids and data-sample partition

Experiment Results

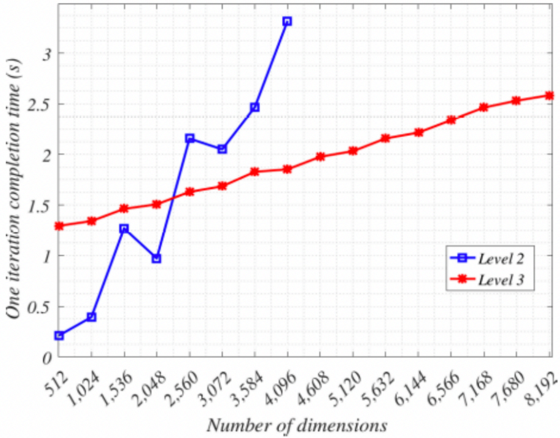

Figure: varying d with 2,000 centroids and 1,265,723 data samples tested on 128 nodes

Experiment Results

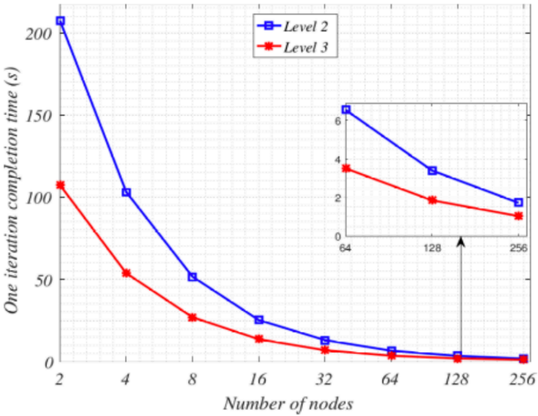

Figure: varying number of nodes used with a xed 4,096 dimension, 2,000 centroids and 1,265,723 data samples

Experiment Results

TABLE 3
Execution time comparison with other architectures

Approaches	Hardware Resources	n	k	d	Execution time per iteration (sec.)	Execution time per iteration by Sunway TaihuLight (sec.)	Max. Speedup
Rossbach, et al [35]	10x NVIDIA Tesla K20M + 20x Intel Xeon E5-2620	1.0E9	120	40	49.4	0.468635 (128 nodes)	105x
Bhimani, et al [4]	NVIDIA Tesla K20M	1.4E6	240	5	1.77	0.025336 (4 nodes)	70x
Jin, et al [26]	NVIDIA Tesla K20c	1.4E5	500	90	5.407	0.110191 (1 node)	49x
Li, et al [29]	Xilinx ZC706	2.1E6	4	4	0.0085	0.002839 (1 node)	3x
Ding, et al [15]	Intel i7-3770K	2.5E6	10,000	68	75.976	2.424517 (16 nodes)	31x

Figure: Execution time comparison with other architectures

Outline

Summary

Contributions of this paper:

1. Level 1 - DataFlow Partition: Store a whole sample and k centroids on single-CPE
2. Level 2 - DataFlow and Centroids Partition: Store a whole sample on single-CPE whilst k centroids on multi-CPE
3. Level 3 - DataFlow, Centroids and Dimensions Partition: Store a whole sample on multi-CPE whilst k centroids on Multi-CG and d dimensions on Multi-CPE
4. The proposed auto-clustering solution is a signicant attempt to support AutoML on a supercomputer system

Thanks

Thanks for your attention!