Large-Scale Hierarchical k-$means$ for Heterogeneous Many-Core Supercomputers

Liandeng Li*§, Teng Yu†, Wenlai Zhao*§, Haohuan Fu*§✉, Chenyu Wang†§, Li Tan‡, Guangwen Yang*§, John Thomson†
*Tsinghua University, China
†University of St Andrews, UK
‡Beijing Technology and Business University, China
§National Supercomputing Center in Wuxi, China
✉*Corresponding Author: haohuan@tsinghua.edu.cn*

***Abstract*—This paper presents a novel design and implementation of k-$means$ clustering algorithm targeting the Sunway TaihuLight supercomputer. We introduce a multi-level parallel partition approach that not only partitions by dataflow and centroid, but also by dimension. Our multi-level (nkd) approach unlocks the potential of the hierarchical parallelism in the SW26010 heterogeneous many-core processor and the system architecture of the supercomputer.**

Our design is able to process large-scale clustering problems with up to 196,608 dimensions and over 160,000 targeting centroids, while maintaining high performance and high scalability, significantly improving the capability of k-$means$ over previous approaches. The evaluation shows our implementation achieves performance of less than 18 seconds per iteration for a large-scale clustering case with 196,608 data dimensions and 2,000 centroids by applying 4,096 nodes (1,064,496 cores) in parallel, making k-$means$ a more feasible solution for complex scenarios.

Keywords: *Supercomputer, Multi/many-core Processors, Clustering, Parallel Computing*

I. Introduction

K-$means$ is a well-known clustering algorithm, used widely in many AI and data mining applications, such as bio-informatics [1], [22], image segmentation [7], [21], information retrieval [35] and remote sensing image analysis [24].

Finding the optimal solution for a general k-$means$ problem is known to be NP-hard [10]. Thus, current high-end k-$means$ applications are limited in terms of the number of dimensions (d), and the number of centroids (k) they can consider, leading to demand for more parallel k-$means$ implementations [2], [24]. Our work will allow k-$means$ data analysis to run at an unprecedented complexity, with significantly higher dimensionality and centroid number than before. Our method is applicable to any problem with an intrinsically high dimensional feature space where traditional dimensionality reduction techniques are commonly used. A typical example in the domain of remote sensing data analysis is shown in section IV, where we manage to process a k-means problem with 4096 dimensions and 7 centroids using 400 MPI processes.

This paper presents a novel method to map data and communication for a multi-level k-$means$ design targeting Sunway TaihuLight, one of the world's fastest supercomputers. This method allows k-$means$ to scale well across a large number of computation nodes, significantly outperforming previously proposed techniques. The proposed implementation is able to process large-scale clustering problems with up to 196,608 dimensions and 160,000 centroids, while maintaining high performance and scalability – a large improvement on previous implementations, as described in Table I. Our method greatly increases the potential scope for k-$means$ applications to solve previously intractable problems.

The key to our approach is a three-level data partition strategy based on hierarchical many-core hardware support. Previous high performance k-$means$ implementations, such as that for the Trinity supercomputer(NNSA) [2] have used a two-level memory approach.

Such an approach, implemented in this paper as *Level 2*, involves partitioning first the number of clusters centroids k by the number of cores in a Core Group(CG) as described in section II, and then by the dataflow n into multiple CGs. Consequently, both n and k are relatively scalable, however each centroid, k, is a d-dimensional vector. The maximum value of $k * d$ is limited by the shared memory of the CG. There are two main drawbacks to this approach: firstly, only one of k or d can be scaled to a large number, as shown in Table I, which details the limits of previous implementations. Secondly, the performance scaling of *Level 2* is shown to be poor as k or d grows towards the high end of possibility for this approach. Therefore even if the memory limits were somehow solved in some other way, the performance scaling would limit the growth of k or d.

These difficulties show the need for a new approach if larger values of k and d are to be reasonably handled. The three-level hierarchical approach proposed in this paper as *Level 3*, addresses both issues of independent growth of k and d, and of scalability.

Level 3 partitions d by the number cores in a CG. The data is further partitioned into multiple CGs by k. Since d and k are partitioned at different hardware levels hierarchically, the total value of $k * d$ is no longer limited by the size of memory available at this level. The dataflow n is then partitioned into new structures - Groups of CG, as shown in Figure 2. In this way, all n, k, d can scale without constraints between each other.

Liandeng Li and Teng Yu are primary authors.
SC18, November 11-16, 2018, Dallas, Texas, USA
978-1-5386-8384-2/18/$31.00

Table I: Parallel k-$means$ implementations

Approaches	Hardware resources	Programming model	Samples n	Clusters k	Dimensions d
General Parallel k-$means$ Implementations					
Böhm, et al [4]	Multi-core Processors	MIMD/SIMD	10^7	40	20
Hadian and Shahrivari [17]	Multi-core Processors	multi-thread	10^9	100	68
Zechner and Granitzer [37]	GPU	CUDA	10^6	128	200
Li, et al [26]	GPU	CUDA	10^7	512	160
Haut, et al [19]	Cloud	OpenStack	10^8	8	58
Cui, et al [8]	Cluster	Hadoop	10^5	100	9
Supercomputer-Oriented k-$means$ Implementations					
Kumar, et al [24]	$Jaguar$, Oak Ridge	MPI	10^{10}	1000	30
Cai, et al [6]	$Gordon$, SDSC	mclappy (parallel R)	10^6	8	8
Bender, et al [2]	$Trinity$, NNSA	OpenMP	370	18	140,256
Our approach	$Sunway$, **Wuxi**	**DMA/MPI**	10^6	**160,000**	**196,608**

This work makes two main contributions:

- The proposed design is the first to allow independent and simultaneous performant variation of dataflow (n), number of centroids (k) and size of dimension (d), using the hierarchical hardware support of the Sunway TaihuLight. The novel partition method allows us to achieve tractable scaling of both k and d to a level higher than achieved before without any interaction constraints. This results in high performance across a wide variation of k and d, are demonstrated by experiments on multiple benchmark workloads with high-dimensional data.
- The proposed three-level partitioning implementation is the first to achieve a flexible supercomputer-based large-scale k-$means$ implementation with varying values of k, d and n. There are no artificial restrictions on small and regular workloads unlike in previous work [2].

The remainder of this paper is presented as follows: Section II describes the background and related work which includes a short description of Sunway supercomputer and the k-$means$ problem definition, the most popular Lloyd algorithm and general parallel implementation, and the state-of-the-art supercomputer-oriented designs in the literature. Section III discusses the three levels scalable design and implementation of k-$means$ on Sunway. Our experimental design and results analysis are given in section IV.

II. Background and related work

A. Sunway TaihuLight and SW26010 Many-Core Processor

Sunway TaihuLight is a world-leading supercomputer, which currently ranks as the second machine in the TOP500 list [28] and achieves a peak performance of 93 petaflops [15].

The high performance and efficiency of Sunway TaihuLight is due to its use of the SW26010 many-core processor. The basic architecture of the SW26010 processor is shown in Figure 1 below. Each processor contains four *core groups (CGs)*. There are 65 cores in each CG, 64 *computing processing element (CPEs)* and a *managing processing element (MPE)*, which are organized as 8 by 8 mesh. The MPE and CPE are both complete 64-bit RISC cores, but they are assigned different tasks while computing. The main function of the MPE is to support the complete interrupt functions, memory management, super-scalar and out-of-order issue/execution, and it is designed for management, task schedule, and data communications. The CPE is assigned to maximize the aggregated computing throughput while minimize the complexity of the micro-architecture.

The SW26010 design differs significantly from the other multi-core and many-core processors: (i) for the memory hierarchy, while the MPE applies a traditional cache hierarchy (32-KB L1 instruction cache, 32-KB L1 data cache, and a 256-KB L2 cache for both instruction and data), each CPE only supplies a 16-KB L1 instruction cache, and depends on a 64 KB *Local directive Memory (LDM)* (also known as *Scratch Pad Memory (SPM)*) as a user-controlled fast buffer. The user-controlled 'cache' leads to some increasing programming difficulties for using fast buffer efficiently, at the same time, providing the opportunity to implement a defined buffering scheme which is beneficial to improve the whole performance in certain cases. (ii) As for the internal information of each CPE mesh, we have a control network, a data transfer network (connecting the CPEs to the memory interface), 8 column communication buses, and 8 row communication buses. The 8 column and row communication buses provide possibility for fast register communication channels to across the 8 by 8 CPE mesh, so users can attain a significant data sharing capability at the CPE level.

B. Related Work

In this section, we provide a formal description of the k-$means$ problem and then present the well-known approach, $Lloyd$ algorithm. Following is a discussion of the general parallel implementations and other supercomputer-based approaches.

1) Problem Definition: The purpose of the k-$means$ clustering algorithm is to find a group of clusters to minimize the mean distances between samples and their nearest centroids. Formalized, given n samples $\mathcal{X}^d = \{x_i^d\} \in \mathbb{R}^d$, $i \in \{1 \dots n\}$, where each sample is a d-dimensional vector $x_i^d = (x_{i1}, \dots, x_{id})$ and we use u to index the dimensions: $u \in \{1 \dots d\}$.

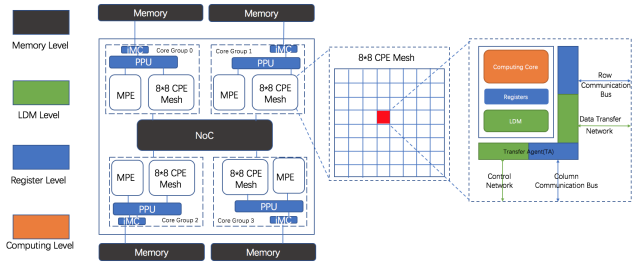

Figure 1: The general architecture of the SW26010 many-core processor

We aim to find k d-dimensional centroids $\mathcal{C}^d = \{c_j^d\} \in \mathbb{R}^d$, $j \in \{1 \dots k\}$ to minimize the object $\mathcal{O}(\mathcal{C})$:

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

end where $a(i)$ is the index of the nearest centroid for sample x_i^d:

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

while $dis(x_i^d, c_j^d)$ is the $Euclidean$ distance between sample x_i^d and centroid c_j^d:

$$dis(x_i^d, c_j^d) = \sum_{u=1}^{d} (x_{iu} - c_{ju})^2$$

2) Lloyd Algorithm: It is well-known that k-means problem is in NP-hard [31]. In the literature, several methods have been proposed to find efficient solutions [5], [9], [13], [30], [31], [34]. While the most popular baseline is still the $Lloyd$ algorithm [29], which is composed by repeating the basic two steps below:

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

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

Note that those notations here are mainly from previous works by Hamerly [18], Newling and Fleuret [30]. We will apply customized notations only when needed. The first step above is to assign each sample into the nearest centroid according to the $Euclidean$ distance. The second step is to update the centroids by moving them to the mean of their assigned samples in the d-dimensional vector space. Those two steps are repeated until each c_j^d is fixed.

3) General Parallel k-means: k-means algorithm has been widely implemented in parallel architectures with shared and distributed memory using either SIMD or MIMD model targeting on multi-core processors [4], [12], [17], GPU-based heterogeneous systems [26], [36], [37], clusters of computer/-cloud [8], [19].

In the parallel case, we use l to index the processors (computing units) $\mathcal{P}$ and use m to denote the total number of processors applied:

$$\mathcal{P} = \{P_l\}, l \in \{1 \dots m\}$$

The dataset $\mathcal{X}^d$ is partitioned uniformly into m processors. Compared against the basic $Lloyd$ algorithm, each processor only assigns a subset ($\frac{n}{m}$) of samples from the original set $\mathcal{X}^d$ before the $Assign$ step. Then the $Assign$ step is finished in parallel by m processors. To formalize the steps, we obtain:

$$1.1 : P_l \leftarrow x_i^d, i \in (1 + (l-1)\frac{n}{m}, l\frac{n}{m})$$

$$1.2 : \forall l \in (1, m), P_l : \ a(i) = arg\ min_{j \in \{1 \dots k\}}\ dis(x_i^d, c_j^d)$$

To facilitate communication between computing units, the Message Passing Interface (MPI) library is mostly applied in common multi-core processor environments. Performance nearly linearly increases with the limited number of processors as the communication cost between processes can be ignored in the non-scalable cases, as demonstrated in [12]. Similarly, the $Update$ steps are finished by m processors in parallel through MPI as well. Processors P_l should communicate with each other before the final c_j^d can be updated. CUDA is applied for implementing those communications when targeting on GPU-based systems [37], Hadoop is used in clusters [8] and OpenStack for cloud architecture [19]. We ignore the formal description of the general reduce-based parallel updating process here because it is not applicable to the proposed methods on our targeted hierarchical many-core processors.

4) Large-scale Parallel k-means on Supercomputers: In addition to general parallel k-means implementations, other customized k-means implementation targeting on supercomputers are more related to our work here.

Kumar, et al [24] implemented the dataflow-partition based parallel k-means on the $Jaguar$, a Cray XT5 supercomputer at Oak Ridge National Laboratory evaluated by real-world geographical datasets. Their implementation applys MPI protocols to achieve broadcasting and reducing and originally scaled the value of k to more than 1,000s level.

Cai, et al [6] designed a similar parallel approach on $Gordon$, a Intel XEON E5 supercomputer at San Diego Supercomputer Center for grouping game players. They applied a parallel R function, $mclapply$, to achieve shared-memory parallelism and test different degree of parallelism by partitioning the original data-flow into different numbers of sets. They did not focus on testing the scalability of their approach but evaluated on the quality of the cluster.

Bender, et al [2] investigated a novel parallel implementation proposed for $Trinity$, the latest National Nuclear Security Administration supercomputer with Intel Knight's Landing processors and their $scratchpad$ two-level memory model. Their approach is the most state-of-the-art comparable work against our proposed methods which can not only partition dataflow, but also partition the number of target clusters k by their $hierarchical$ two-level memory support - cache associated with each core and $scratchpad$ for share. Adapted originally from [16], their partitioning algorithm partitioned the input dataset into $\frac{nd}{M}$ sets, where M is the size of the $scratchpad$, and then reduced $k\frac{nd}{M}$ centroids recursively if needed. Based on this partition, their approach scaled d into 100,000s level.

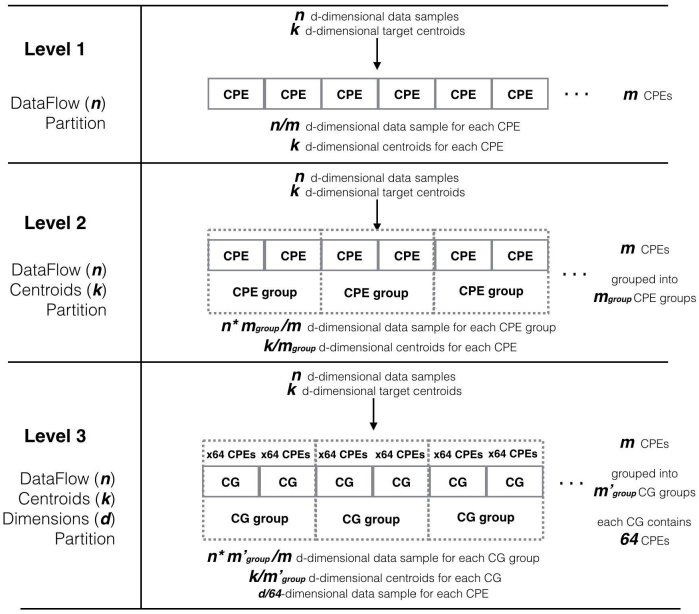

Figure 2: Three-level k-$means$ design for data partition and parallelism on Sunway architecture

A fundamental bottleneck in their approach is that based on only two-level memory, it is still impossible to partition and then scale both k and d independently. This leads to the interaction constraint between k and d as discussed in their paper:

$$Z < kd < M$$

where Z is the size of cache. This partition-based method is not efficient if all k centroids could fit into one cache. In practice, this limits the value of k to be less than 18 and d to be greater than 152,917 in their experiments. We claim that our proposed approach with underlining data partitioning methods based on hierarchical many-core processors achieves the needed multi-level fully nkd partition with architectural support to thoroughly solve this bottleneck.

We formalize the background work of both general parallel k-$means$ and supercomputer-oriented implementations as shown in Table I.

III. Multi-level large-scale k-$means$ design

The scalability and performance of parallel k-$means$ algorithm on large-scale heterogeneous systems and supercomputers are mainly bounded by the memory and bandwidth. To achieve efficient large-scale k-$means$ on the Sunway supercomputer, we explore the hierarchical parallelism on our heterogeneous many-core architecture. We demonstrate the proposed scalable methods on three parallelism levels by how we partition the data.

- Level 1 - $DataFlow$ Partition: Store a whole sample and k centroids on single-CPE
- Level 2 - $DataFlow$ and $Centroids$ Partition: Store a whole sample on single-CPE whilst k centroids on multi-CPE
- 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

An abstract graph of how we partition the data into multiple levels is presented in Figure 2.

A. Level 1 - DataFlow Partition

Algorithm 1 Basic Parallel k-$means$

1: **INPUT:** Input dataset $\mathcal{X} = \{x_i | x_i \in \mathbb{R}^d, i \in [1, n]\}$, and initial centroid set $\mathcal{C} = \{c_j | c_j \in \mathbb{R}^d, j \in [1, k]\}$
2: $P_l \xleftarrow{load} \mathcal{C},\ l \in \{1 \dots m\}$
3: **repeat**
4: // Parallel execution on all CPEs:
5: **for** $l = 1$ *to* m **do**
6: Init a local centroids set $\mathcal{C}^l = \{c_j^l | c_j^l = \mathbf{0}, j \in [1, k]\}$
7: Init a local counter $count^l = \{count_j^l | count_j^l = 0, j \in [1, k]\}$
8: **for** $i = (1 + (l-1) * \frac{n}{m})$ *to* $(l * \frac{n}{m})$ **do**
9: $P_l \xleftarrow{load} x_i$
10: $a(i) = arg\ min_{j \in \{1 \dots k\}} dis(x_i, c_j)$
11: $c_{a(i)}^l = c_{a(i)}^l + x_i$
12: $count_{a(i)}^l = count_{a(i)}^l + 1$
13: **for** j = 1 *to* k **do**
14: AllReduce c_j^l and $count_j^l$
15: $c_j^l = \frac{c_j^l}{count_j^l}$
16: **until** $\mathcal{C}^l == \mathcal{C}$
17: **OUTPUT:** $\mathcal{C}$

In the simple case, we run the first step, $Assign$, on each CPE in parallel while using multi-CPE collaboration to implement the second step, $Update$. The pseudo code of this case is shown in Algorithm 1.

The $Assign$ step is implemented similarly to the traditional parallel k-$means$ algorithm – (1.1) and (1.2) as above. Given n samples, we partition into multiple CPEs. Each CPE (P_l) firstly reads one sample x_i and finds the minimum distances dis from itself to all centroids c_j to obtain $a(i)$. Then two variables are accumulated for each cluster centroid c_j according to $a(i)$, shown in line 11 and 12. The first variable stores the vector sum of all the samples assigned to c_j, notated as $c_{a(i)}^l$. The second variable counts the total number of samples assigned to c_j, notated as $count_{a(i)}^l$.

In the $Update$ step, we first accumulate the c_j^l and $count_j^l$ of all CPEs by performing two AllReduce operations, so that all CPEs can obtain the assignment results of the whole input dataset. We use $register\ communication$ [14] to implement intra-CG AllReduce operation and use MPI_AllReduce for inter-CG AllReduce. After the accumulation, the $Update$ step is performed to calculate new centroids, as shown in line 15.

Analysis: Considering a one-CG task, we analyse the constraints on scalability in terms of memory limitation of each CPE. Based on the steps above, one CPE has to accommodate at least one sample x_i, all cluster centroids $\mathcal{C}$, k centroids'

accumulated vector sum $\mathcal{C}^l$ and k centroids' counters $count^l$. Considering that each CPE has a limited size of LDM, we obtain the constraint ($\mathbf{C}_1$) below:

$$\mathbf{C}_1: \quad d(1+k+k)+k \leq LDM$$

Since both the number of centroids k and the dimension d for each sample x_i should at least be 1, we obtain two more boundary constraints ($\mathbf{C}_2$) and ($\mathbf{C}_3$) below, separately:

$$\mathbf{C}_2: \quad 3d+1 \leq LDM$$

$$\mathbf{C}_3: \quad 3k+1 \leq LDM$$

Now we analyse the performance under bandwidth bounds. Note that the $Assign$ step of computing $a(i)$ for each sample x_i is completed fully in parallel on the m CPEs. Given the bandwidth of multi-CPE architecture to be B, the DMA time of reading data from main memory can be simply formalized as:

$$\mathbf{T}_{read}: \quad (\frac{n*d}{m}+k*d)/B$$

Theoretically, a linear speedup for computing time to at most n times against the serial implementation can be obtained for the $Assign$ step if we can apply $m=n$ CPEs in total.

The two AllReduce operations are the bottleneck process in the $Update$ step. The $register\ communication$ technique for internal multi-CPE communication guarantees a high-performance with a normally 3x to 4x speedup than other on-chip and Internet communication techniques (such as DMA and MPI) for this bottleneck process (referring to the experimental configuration section for detailed quantitative values). Given the bandwidth of $register\ communication$ to be R, the time for the AllReduce process can be formalized as:

$$\mathbf{T}_{comm}: \quad \frac{n}{m}((1+k)*d)/R$$

B. Level 2 - DataFlow and Centroids Partition

To scale the number of k for cluster centroids $\mathcal{C}$, we use multiple (up to 64) CPEs in one CG to partition the set of centroids. The number of CPEs grouped to partition the centroids is denoted by m_{group}. For illustration, we use l' to index the CPE groups {P}. Then we have:

$$\{P\}_{l'} := \{P_l\},\ l \in (1+(l'-1)*m_{group}, l'*m_{group})$$

The pseudo code of this case is shown in Algorithm 2. To partition k centroids on m_{group} CPEs, we need to do a new sub-step against the previous case as shown in line 2. Then different from the $Assign$ step in above case, we partition each data sample x_i in each CPE group as shown in line 8. After that, similar to (1.2), all P_l in each $\{P\}_{l'}$ can still compute a partial value of $a(i)$ (named as $a(i)'$) fully in parallel without communication. Note that the domain of j in line 11 is only a subset of $(1,\ldots,k)$ as presented above in line 2, so we need to do one more step by data communication between CPEs in each CPE group to obtain the final a(i) as shown in line 10.

Then the $Update$ step is similar to previous case. We just view one CPE group as one basic computing unit, which conducts what a CPE did in the previous case. Each CPE only computes values of subset of centroids $\mathcal{C}$ and does not need further communications in this step as it only needs to store this subset.

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

1: **INPUT:** Input dataset $\mathcal{X}=\{x_i|x_i \in \mathbb{R}^d, i \in [1,n]\}$, and initial centroid set $\mathcal{C}=\{c_j|c_j \in \mathbb{R}^d, j \in [1,k]\}$
2: $P_l \xleftarrow{load} c_j \quad j \in (1+mod(\frac{l-1}{m_{group}})*\frac{k}{m_{group}},\ (mod(\frac{l-1}{m_{group}})+1)*\frac{k}{m_{group}})$
3: **repeat**
4: // Parallel execution on each CPE group $\{P\}_{l'}$:
5: **for** $l'=1\ to\ \frac{m}{m_{group}}$ **do**
6: Init a local centroids set $\mathcal{C}^{l'}$ and counter $count^{l'}$
7: **for** i = $(1+(l'-1)\frac{n*m_{group}}{m})\ to\ (l'\frac{n*m_{group}}{m})$ **do**
8: $\{P\}_{l'} \xleftarrow{load} x_i$
9: $a(i)' = arg\ min_j\ dis(x_i, c_j)$
10: $a(i) = min.\ a(i)'$
11: $c^{l'}_{a(i)} = c^{l'}_{a(i)} + x_i$
12: $count^{l'}_{a(i)} = count^{l'}_{a(i)} + 1$
13: **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**
14: AllReduce $c^{l'}_j$ and $count^{l'}_j$
15: $c^{l'}_j = \frac{c^{l'}_j}{count^{l'}_j}$
16: **until** $\cup\ \mathcal{C}^{l'} == \mathcal{C}$
17: **OUTPUT:** $\mathcal{C}$

Analysis: To analyse the scalability of k in this case, the amount of original k centroids distributed in m_{group} CPEs leads to a easier constraint of k against the ($\mathbf{C}_3$) above:

$$\mathbf{C'}_3: \ 3k+1 \leq m_{group}\ *LDM\ (m_{group} \leq 64)$$

Based on this, we can also easily scale the ($\mathbf{C}_1$) as follow:

$$\mathbf{C'}_1: \ d(1+k+k)+k \leq m_{group}\ *LDM\ (m_{group} \leq 64)$$

Note that we still need to accommodate at least one d-dimensional sample in one CPE, so the ($\mathbf{C}_2$) should be kept as before: $\mathbf{C'}_2 := \mathbf{C}_2$

As for performance, since m_{group} CPEs in one group should read the same sample simultaneously, the processors need more time to read the input data samples than the first case, but only partial cluster centroids need to be read by each CPE:

$$\mathbf{T'}_{read}: \quad (\frac{n*d*m_{group}}{m}+\frac{k}{m_{group}}*d)/B$$

As for the data communication needed, there is one more bottleneck process (line 12) than before. Comparing against the above cases, multiple CPE groups can be allocated in different processors. Those communication need to be done through MPI which is much slower than internal processor multi-CPEs $register\ communication$. Given the bandwidth of network communication through MPI to be M, we obtain:

$$\mathbf{T'}_{comm}: \frac{k}{m_{group}}/R+\frac{n*m_{group}}{m}((1+k)*d))/M$$

C. Level 3 - DataFlow and Centroids and Dimensions Partition

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

1: **INPUT:** Input dataset $\mathcal{X} = \{x_i | x_i \in \mathbb{R}^d, i \in [1, n]\}$, and initial centroid set $\mathcal{C} = \{c_j | c_j \in \mathbb{R}^d, j \in [1, k]\}$
2: $CG_{l''} \xleftarrow{load} c_j^d,\ l'' \in \{1 \dots \frac{m}{64}\}, j \in (1 + mod(\frac{l''-1}{m'_{group}}) * \frac{k}{m'_{group}},\ (mod(\frac{l''-1}{m'_{group}}) + 1) * \frac{k}{m'_{group}})$
3: **repeat**
4: // Parallel execution on each CG group $\{CG\}_{l''}$:
5: **for** $l'' = 1$ *to* $\frac{m}{64}$ **do**
6: Init a local centroids set $\mathcal{C}^{l''}$ and counter $count^{l''}$
7: **for** i = $(1 + (l''-1)\frac{n*m'_{group}}{m})$ *to* $(l''\frac{n*m'_{group}}{m})$ **do**
8: **for** u = $(1+mod(\frac{l-1}{64}) * \frac{d}{64}$ *to* $(mod(\frac{l-1}{64})+1) * \frac{d}{64})$ **do**
9: $CG_{l''} \leftarrow x_i\ (P_l \leftarrow x_i^u)$
10: $a(i)' = arg\ min_j\ dis(x_i, c_j)$
11: $a(i) = min.\ a(i)'$
12: $c_{a(i)}^{l''} = c_{a(i)}^{l''} + x_i$
13: $count_{a(i)}^{l''} = count_{a(i)}^{l''} + 1$
14: **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**
15: AllReduce $c_j^{l''}$ and $count_j^{l''}$
16: $c_j^{l''} = \frac{c_j^{l''}}{count_j^{l''}}$
17: **until** $\cup\ \mathcal{C}^{l''} == \mathcal{C}$
18: **OUTPUT:** $\mathcal{C}$

To scale the number of dimension d for each sample x_i and further scale k, we store and partition one d-dimensional sample by one CG with 64 CPEs and then implement the algorithm on multiple CGs. The pseudo code of this case is shown in Algorithm 3.

Recall we use u to index the data dimension: $u \in (1 \dots d)$; Now we use l'' to index the CGs and m'_{group} to denote the number of CGs grouped together to partition k centroids. Consider that we apply m CPEs in total and each CG contains 64 CPEs, then we have $l'' \in (1, \dots, \frac{m}{64})$, $m'_{group} \leq \frac{m}{64}$ and:

$$CG_{l''} := \{P_l\},\ l \in (1 + 64(l''-1),\ 64l'')$$

To partition k centroids on multiple CGs, we obtain an updated step against the previous case as shown in line 2. To partition each d-dimensional sample x_i^d on 64 CPEs in one CG, we obtain the following step as shown in line 9.

Similar to the above case, all $CG_{l''}$ in each CG group compute the partial value $a(i)'$ fully in parallel and then communicate to obtain the final $a(i)$. Multi-CG communication in multiple many-core processors (nodes) is implemented through MPI interface. Then the $Update$ step is also similar to the previous case. Now we view one CG as one basic computing unit which conducts what one CPE did before and we view what a CG group does as what a CPE group did before.

Analysis: In this case, each CG with 64 CPEs accommodates one d-dimensional sample x_i. Then we can scale the previous $(\mathbf{C}_2)$ as follow:

$$\mathbf{C''}_2 :\ 3d + 1 \leq 64 * LDM$$

Consider we use totally m'_{group} CGs to accommodate k centroids in this case, then $(\mathbf{C}_3)$ will scale as follow:

$$\mathbf{C''}_3 :\ 3k + 1 \leq m'_{group} * 64 * LDM$$

Note that the domain of m'_{group} seems limited by the total number of CPEs applied, m. But in fact, this number can be large-scale as we target on the supercomputer with tens of millions of cores. Finally, $(\mathbf{C}_1)$ will scale as follow:

$$\mathbf{C''}_1 :\ d(1 + k + k) + k \leq 64 * m'_{group} * LDM$$

which is equal to:

$$\mathbf{C''}_1 :\ d(1 + k + k) + k \leq m * LDM$$

$\mathbf{C''}_1$ is the breakthrough contribution over other state-of-the-art work [2]: the total amount of $d * k$ is not limited by a single or shared memory size any more. It is fully scalable by the total number of processors applied (m). In a modern supercomputer, this value can be large-scaled up-to tens of millions when needed.

Considering performance, note that m'_{group} CGs (64 CPEs in each) in one group should read the same sample simultaneously. In another aspect, each CPE only needs to read a partial of the given d-dimension of original data sample together with a partial of k centroids similarly as before, then we obtain a similar reading time:

$$\mathbf{T''}_{read} :\quad (\frac{n * d * m'_{group}}{m} + \frac{k}{m'_{group}} * \frac{d}{64})/B$$

Comparing against the above cases, multiple CGs in CG groups allocated in different many-core processors need communication to update centroids through MPI. Given the bandwidth of network communication through MPI to be M, the cost between multiple CG groups can be formalized as:

$$\mathbf{T''}_{comm} : (\frac{k}{m'_{group}} + \frac{n * m'_{group}}{m}((1 + k) * d))/M$$

The network architecture of Sunway TaihuLight is a two-level fat tree. 256 computing nodes are connected via a customized inter-connection board, forming a *super-node*. All super-nodes are connected with a central routing server. The intra super-node communication is more efficient than the inter super-node communication. Therefore, in order to improve the overall communication efficiency of our design, we should make a CG group located within a super-node if possible.

D. Impact of Multi-level Large-scale Design

As described in the background section, *Level 1* is based on the well-researched parallel k-$means$ deign using dataflow partition (n-partition) which has been implemented on other supercomputers including $Jaguar$ [24] and $Gordon$ [6] to process regular big dataset with up to 1,000s centroids and small

Table II: Benchmarks from UCI and ImgNet

Data Set	n	k	d
Kegg Network	6.5E4	256	28
Road Network	4.3E5	10,000	4
US Census 1990	2.5E6	10,000	68
ILSVRC2012 (ImgNet)	1.3E6	160,000	196,608

number of dimensions. *Level 2* provides similar functionality to Bender et al. [2] approach targeting on $Trinity$, implementing both dataflow and centroids partition (nk-partition) to successfully handle big dataset with large-scale dimensions. *Level 3* is our original first ever approach to finally achieve all dataflow, centroids and data samples (nkd-partition) simultaneously to successfully handle big dataset with both large-scale dimensions and large-scale centroids achieving high performance.

The multi-level large-scale approaches together also give us the needed flexibility to handle both high dimensional and low dimensional dataset efficiently on supercomputer, which also breaks the limitation in current state-of-the-art design by Bender et al. [2] which claims only efficient for dataset with larger than 100,000 dimensions.

IV. Experimental Evaluation

We describe our experimental evaluation in this section. We run our proposed methods on Sunway TaihuLight, with a hierarchical SW26010 many-core processor as the main processor. architecture.

We first describe the datasets applied and then discuss the experimental metrics. The experimental strategy is presented followed by the results on the three scalable levels, a comparison between partitioning strategies, and analysis.

A. Experimental Datasets

The datasets we applied in experiments come from well-known benchmark suites including UCI Machine Learning Repository [32] and ImgNet [20]. We briefly present the datasets in Table II, where the first three normal size benchmarks (*Kegg Network, Road Network, US Census 1990*) are from UCI and the final high-dimensional benchmarks (*ILSVRC2012*) are from ImgNet.

We do not describe the more detailed technical and background descriptions of those benchmarks as they are well-known and commonly applied in the literature.

B. Experimental Design and Metrics

The experiments have been conducted to demonstrate scalability, high performance and flexibility by increasing the number of centroids k and number of dimensions d on multiple benchmarks with vary data size n. The three-level designs are tested targeting different benchmarks. Different hardware setup will be provided for testing different scalable levels:

- *Level 1* - One SW26010 many-core processor is applied, which contains 256 64-bit RISC CPEs running at 1.45 GHz, grouped in 4 CGs in total. As 64 KB LDM buffer is associated with each CPE and 32 GB DDR3 memory is shared for the 4 CGs, we setup 16 MB LDM and 32 GB DDR3 memory support in total. The theoretical memory bandwidth for register communication is 46.4 GB/s and for DMA is 32 GB/s.
- *Level 2* - Up-to 256 SW26010 many-core processors are applied, which contains 65,536 64-bit RISC CPEs running at 1.45 GHz, grouped in 1,024 CGs in total. We setup 4 GB LDM and 8 TB DDR3 memory support in total. The theoretical memory bandwidth for register communication is 46.4 GB/s and for DMA is 32 GB/s. The bidirectional peak bandwidth of the network between multiple processors is 16 GB/s.
- *Level 3* - Up-to 4,096 SW26010 many-core processors are applied, which contains 1,064,496 64-bit RISC cores running at 1.45 GHz, grouped in 16,384 CGs in total. In this setup, 64 GB LDM and 128 TB DDR3 memory are supported in total. The bidirectional peak bandwidth of the network between multiple processors is 16 GB/s.

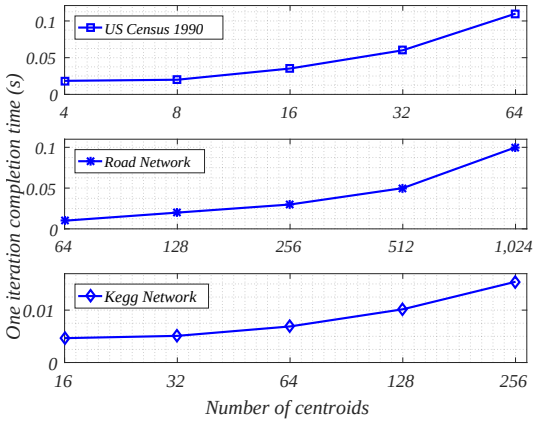

Figure 3: Level 1 - dataflow partition

The main performance metric we are concerned with here is *one iteration completion time*. Note that the total number of iterations needed and the quality of the solution (precision) are not considered in our experiments as our work does not relate to the optimization of the underlining Lloyd algorithm or the solution of k-$means$ algorithm.

C. Experimental Results and Analysis

We report the results of three different partition strategies: *Level 1* – a baseline single-level partition strategy, *Level 2* – an implementation of a state-of-the-art two-level partition strategy used in recent supercomputer implementations [2], and *Level 3* – our novel three-level partition strategy.

Since each partitioning strategy is only able to run successfully at certain ranges of k and d, it is not possible to compare them directly across the whole range benchmarks as the benchmarks have limits in terms of dataset size. For this reason, we first evaluate each strategy independently on the most suitable benchmarks for the strategy in question to show how each performs in the range for which they are most

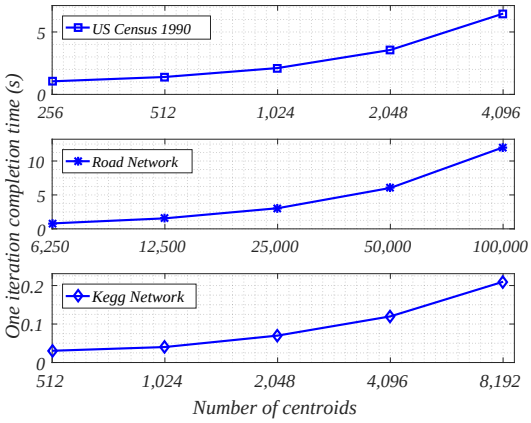

Figure 4: Level 2 - dataflow and centroids partition

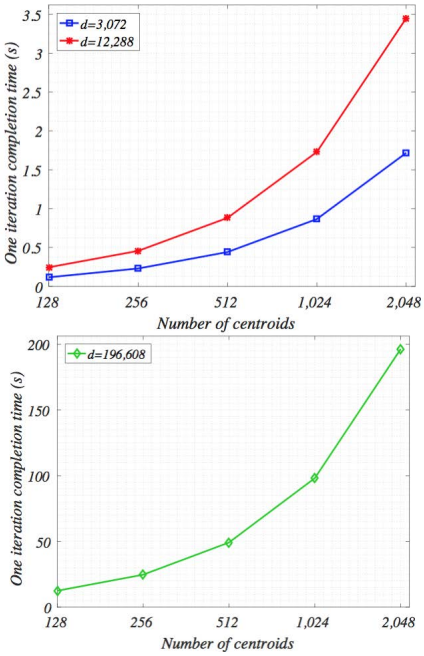

Figure 5: Level 3 - dataflow, centroids and data-sample partition

suited. The second part of our evaluation compares the partition strategies directly on benchmarks where the possible range of k and d overlap. This shows how our proposed *Level 3* strategy scales significantly better than *Level 2* over varying k, d, and number of computational nodes.

1) Level 1 - dataflow partition: The *Level 1* (n-partition) parallel design is applied to three UCI datasets (*US Census 1990, Road Network, Kegg Network*) with their original sizes (n = 2,458,285, 434,874 and 65,554 separately) and data dimensions (d = 68, 4 and 28) for cross number of target centroids (k). The purpose of these experiments is to demonstrate the efficiency and flexibility of this approach on datasets with relatively low size, dimensions and centroid values. Figure 3 shows the *one iteration completion time* for those datasets over increasing number of clusters, k. As the number of k increases, the completion time on this approach grows linearly.

2) Level 2 - dataflow and centroids partition: The level 2 (nk-partition) parallel design is applied to same three UCI datasets as above, but for a large range of target centroids (k). The purpose of these experiments is to demonstrate the efficiency and flexibility of the proposed approaches on datasets with large-scale target centroids (less than 100,000). Figure 4 shows the *one iteration completion time* of the three datasets of increasing number of clusters, k. As the number of k increasing, the completion time from this approach grows linearly. We conclude that this approach works well when one dimension is varied up to the limits previously published.

3) Level 3 - dataflow, centroids and dimensions partition: The *Level 3* (nkd-partition) parallel design is applied to a subset of ImgNet datasets (*ILSVRC2012*) with its original size (n = 1,265,723). The results are presented with varying number of target centroids (k) and data dimension size (d) with an extremely large domain. We also test the scalability varying the number of computational nodes. The purpose of these experiments is to demonstrate the high performance and scalability of the proposed approaches on datasets with large size, extremely high dimensions and target centroids. Figure 5 shows the completion time of the dataset of increasing number of clusters, k = 128, 256, 512, 1024 and 2,048 with increasing number of dimensions, d = 3,072 (32*32*3), 12,288 (64*64*3) and 196,608 (256*256*3).

To further investigate the scalability of our approach, we test two more cases by either further scaling centroids by certain number of data dimensions (d = 3,072) and number of nodes ($nodes$ = 128) or further scaling nodes applied by certain number of data dimensions (d = 196,608) and number of centroids (k = 2,000). The results of those two tests are shown in Figure 6.

As both k and d increase, the completion time from our approach continues to scale well, demonstrating our claimed high performance and breakthrough large scalability.

4) Comparison of partition levels: In this section we experimentally compare the *Level 2* approach with *Level 3*.

Figure 7 shows how *one iteration completion time* grows as the number of dimensions increases. The Level 2 approach outperforms Level 3 when the number of dimensions is relatively small. However, the Level 3 approach scales significantly better with growing dimensionality, outperforming Level 2 for all d greater than 2560. The Level 2 approach cannot run with d greater than 4096 in this scenario due to memory constraints. However, it is clear that, even if this problem were solved, the poor scaling would still limit this approach. The completion time for Level 2 falls twice unexpectedly between 1536 and 2048, and between 2560 and 3072. This is due to the crossing of communication boundaries in the architecture of the supercomputer – the trend remains clear however.

Figure 8 shows how the *one iteration completion time* grows as the number of centroids, k increases. Since the number of d is fixed at 4096, the Level 3 approach actually always outperforms Level 2, with the gap increasing as k increases.

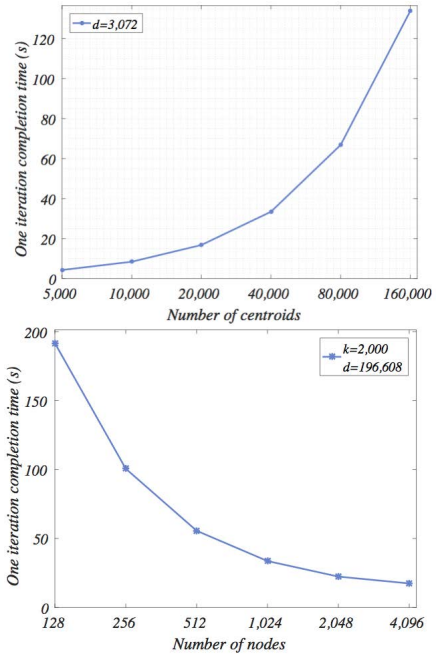

Figure 6: Level 3 - large-scale on centroids and nodes

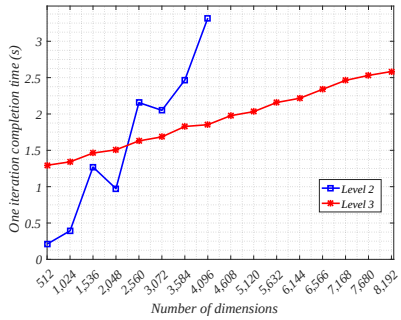

Figure 7: Comparison: varying d with 2,000 centroids and 1,265,723 data samples tested on 128 nodes

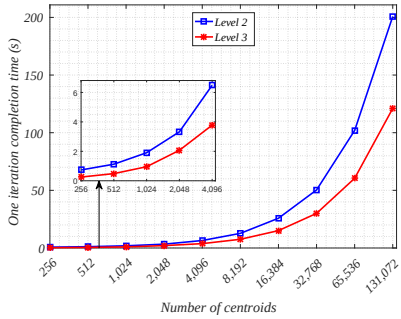

Figure 8: Comparison test: varying k with 4,096 dimensions and 1,265,723 data samples tested on 128 nodes

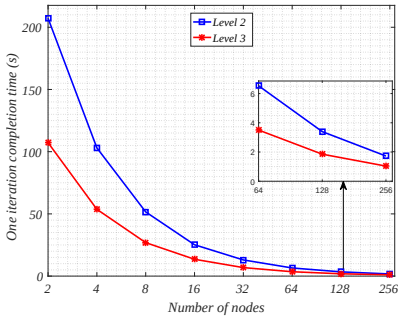

Figure 9: Comparison test: varying number of nodes used with a fixed 4,096 dimension, 2,000 centroids and 1,265,723 data samples

This scaling trend is replicated at lower levels of d too, though Level 2 initially outperforming Level 3 at lower values of k.

Figure 9 shows how both Level 2 and Level 3 scale across an increasing number of computation nodes. Level 3 clearly outperforms Level 2 in all scenarios. The values of k and d are fixed, as described in the graph caption, at levels which Level 2 can operate. The performance gap narrows as more nodes are added, but remains significant. Clearly the exact performance numbers will vary with other values k and d, as can be inferred from other results, but the main conclusion we draw here is that Level 3 generally scales well.

5) Comparison with other architectures: As discussed, state-of-the-art supercomputing-oriented approaches are tested either on their specific datasets [6], [24] or publish only their relative speedups [2] instead of execution times. It is not possible to compare our actual execution time with these supercomputing-oriented approaches directly. Additionally, wallclock execution times are problematic to compare across vastly differing architectures with different budgets.

To give some insight into the performance we obtain, we compare execution time with other architectures directly where this is possible. We present five comparable results from published literature in Table III. Based on the differing workload sizes presented in these papers, we adjust the hardware configuration for Sunway TaihuLight, changing the number of nodes utilized. This is determined by the size of the task in terms of k and d where no further performance gains are possible by adding more nodes. The number of nodes varies from just one node for a single processing unit [23], [27] to 128 nodes in [33].

We report results against a heterogeneous node based approach running a custom implementation of parallel k-means on ten heterogeneous nodes, each node consisting of an NVIDIA Tesla K20M GPU with two Intel Xeon E5-2620 CPUs [33]. Further, we compare against two GPU based implementations running on an NVIDIA Tesla K20M GPU and an NVIDIA Tesla K20c GPU respectively [3], [23], an FPGA based approach running a custom parallel k-means implementation on Xilinx ZC706 FPGA [27], and a multi-core processor based approach running a custom implementation of parallel k-means on 8-core Intel i7-3770k processor [13].

Table III: 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.)	Speedup
Rossbach, et al [33]	10x NVIDIA Tesla K20M + 20x Intel Xeon E5-2620	1.0E9	120	40	49.4	0.468635 (128 nodes)	105x
Bhimani, et al [3]	NVIDIA Tesla K20M	1.4E6	240	5	1.77	0.025336 (4 nodes)	70x
Jin, et al [23]	NVIDIA Tesla K20c	1.4E5	500	90	5.407	0.110191 (1 node)	49x
Li, et al [27]	Xilinx ZC706	2.1E6	4	4	0.0085	0.002839 (1 node)	3x
Ding, et al [13]	Intel i7-3770K	2.5E6	10,000	68	75.976	2.424517 (16 nodes)	31x

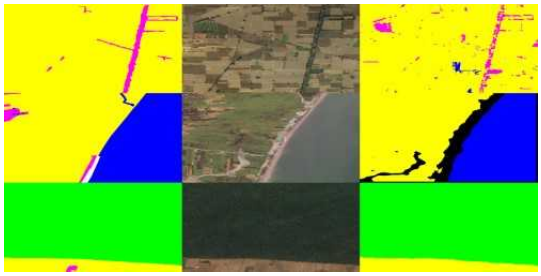

Figure 10: Remote Sensing Image Classification. Left: the result from baseline approach provided by [11]. Middle: the corresponding original image. Right: our classification result.

The proposed approach running on the Sunway TaihuLight supercomputer achieves more than 100x speedup over the high-performance heterogeneous nodes based approach, between 50x-70x speedup than those single GPU based approaches, and 31x speedup over multi-core CPU based approach on their largest solvable workload sizes.

D. Impact on Applications

As a widely used clustering algorithm, a highly efficient and scalable k-$means$ implementation is important to support applications with increasingly large problem sizes and data processing requirements. To demonstrate the efficacy of our design on a real application, we report results for the *land cover classification* application. This is a popular remote sensing problem, requiring unsupervised methods to handle high numbers of unlabeled remote sensing images [25].

K-$means$ has already been used for regional land cover classification with small number of targeted classes. For example, Figure 10 shows our result of classifying a remote sensing image (from a public dataset called Deep Globe 2018 [11]) into 7 classes, representing the urban, the agriculture, the rangeland, the forest, the water, the barren and unknown. There are 803 images in the Deep Globe 2018 dataset, and each image has about $2k \times 2k$ pixels. The resolution of the image is 50cm/pixel. In this problem definition, we cluster on one image, where n is 5838480, k is 7 and d is 4096, which can be done with 400 SW26010 many-core processors. Our Level 3 design can process the clustering dataset efficiently. In recent years, high-resolution remote sensing images have become more common in land cover classification problems. The problem definition on high-resolution images is more complex as the classification sample can be a block of pixels instead of one pixel, which means the d can be even larger.

Real world research of high-resolution land cover classification and other similar problems are currently in progress on the Sunway TaihuLight supercomputer, using the method proposed in this paper. Further significant applications with intrinsic high dimensionality are potentially supported.

V. Conclusions

This paper presents the first ever fully data partitioned (nkd-partition) approach for parallel k-$means$ implementation to achieve scalability and high performance at large numbers of centroids and high data dimensionality simultaneously. Running on the Sunway TaihuLight supercomputer, it breaks previous limitations for high performance parallel k-$means$, allowing data scientists a new powerful tool with great potential.

The proposed multi-level approach also achieves greater flexibility on general workloads with varying data size, target centroids and data dimensions compared with previous supercomputer-specific approaches. The novel design unlocks the potential of hierarchical hardware support of the Sunway TaihuLight for k-$means$, and shows how to optimize this and potentially similar algorithms for a cutting edge heterogeneous many-core supercomputer design.

Acknowledgement

H.Fu is supported by National Key R&D Program of China (Grant No. 2017YFA0604500), by National Natural Science Foundation of China (Grant No. 91530323, 5171101179).

L.Li is supported by by the National Natural Science Foundation of China (Grant No. 61702297), by China Postdoctoral Science Foundation (grant no. 2016M601031).

J.Thomson and T.Yu are supported by the EPSRC grants "Discovery" EP/P020631/1, "ABC: Adaptive Brokerage for the Cloud" EP/R010528/1, and EU Horizon 2020 grant TeamPlay: "Time, Energy and security Analysis for Multi/Many-core heterogenous PLAtforms" (ICT-779882, https://teamplay-h2020.eu).

L.Tan and C.Wang is supported by the National Natural Science Foundation of China (Grant No. 61702020) and its special supporting fund (PXM2018_014213_000033), by Beijing Natural Science Foundation (Grant No. 4172013).

G.Yang is supported by the National Key R&D Program of China (Grant No. 2016YFA0602200).

W.Zhao is supported by the National Key R&D Program of China (Grant No. 2017YFB0202204).

REFERENCES

[1] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of computational biology*, 6(3-4):281–297, 1999.

[2] Michael A Bender, Jonathan Berry, Simon D Hammond, Branden Moore, Benjamin Moseley, and Cynthia A Phillips. k-means clustering on two-level memory systems. In *Proceedings of the 2015 International Symposium on Memory Systems*, pages 197–205. ACM, 2015.

[3] Janki Bhimani, Miriam Leeser, and Ningfang Mi. Accelerating k-means clustering with parallel implementations and gpu computing. In *High Performance Extreme Computing Conference (HPEC), 2015 IEEE*, pages 1–6. IEEE, 2015.

[4] Christian Böhm, Martin Perdacher, and Claudia Plant. Multi-core k-means. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 273–281. SIAM, 2017.

[5] Thomas Bottesch, Thomas Bühler, and Markus Kächele. Speeding up k-means by approximating euclidean distances via block vectors. In *International Conference on Machine Learning*, pages 2578–2586, 2016.

[6] Y Dora Cai, Rabindra Robby Ratan, Cuihua Shen, and Jay Alameda. Grouping game players using parallelized k-means on supercomputers. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, page 10. ACM, 2015.

[7] Guy Barrett Coleman and Harry C Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.

[8] Xiaoli Cui, Pingfei Zhu, Xin Yang, Keqiu Li, and Changqing Ji. Optimized big data k-means clustering using mapreduce. *The Journal of Supercomputing*, 70(3):1249–1259, 2014.

[9] Ryan R Curtin. A dual-tree algorithm for fast k-means clustering with large k. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 300–308. SIAM, 2017.

[10] Sanjoy Dasgupta. *The hardness of k-means clustering*. Department of Computer Science and Engineering, University of California, San Diego, 2008.

[11] Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raskar. Deepglobe 2018: A challenge to parse the earth through satellite images. *ArXiv e-prints*, 2018.

[12] Inderjit S Dhillon and Dharmendra S Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Large-scale parallel data mining*, pages 245–260. Springer, 2002.

[13] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In *International Conference on Machine Learning*, pages 579–587, 2015.

[14] Jiarui Fang, Haohuan Fu, Wenlai Zhao, Bingwei Chen, Weijie Zheng, and Guangwen Yang. swdnn: A library for accelerating deep learning applications on sunway taihulight. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 615–624. IEEE, 2017.

[15] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. The sunway taihulight supercomputer: system and applications. *Science China Information Sciences*, 59(7):072001, 2016.

[16] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE transactions on knowledge and data engineering*, 15(3):515–528, 2003.

[17] Ali Hadian and Saeed Shahrivari. High performance parallel k-means clustering for disk-resident datasets on multi-core cpus. *The Journal of Supercomputing*, 69(2):845–863, 2014.

[18] Greg Hamerly. Making k-means even faster. In *Proceedings of the 2010 SIAM international conference on data mining*, pages 130–140. SIAM, 2010.

[19] Juan Mario Haut, Mercedes Paoletti, Javier Plaza, and Antonio Plaza. Cloud implementation of the k-means algorithm for hyperspectral image analysis. *The Journal of Supercomputing*, 73(1):514–529, 2017.

[20] ImgNet ILSVRC2012. http://www.image-net.org/challenges/lsvrc/2012/.

[21] Anil K Jain and Richard C Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[22] Daxin Jiang, Chun Tang, and Aidong Zhang. Cluster analysis for gene expression data: a survey. *IEEE Transactions on knowledge and data engineering*, 16(11):1370–1386, 2004.

[23] Yu Jin and Joseph F Jaja. A high performance implementation of spectral clustering on cpu-gpu platforms. *arXiv preprint arXiv:1802.04450*, 2018.

[24] Jitendra Kumar, Richard T Mills, Forrest M Hoffman, and William W Hargrove. Parallel k-means clustering for quantitative ecoregion delineation using large data sets. *Procedia Computer Science*, 4:1602–1611, 2011.

[25] Weijia Li, Haohuan Fu, Le Yu, Peng Gong, Duole Feng, Congcong Li, and Nicholas Clinton. Stacked autoencoder-based deep learning for remote-sensing image classification: a case study of african land-cover mapping. *International Journal of Remote Sensing*, 37(23):5632–5646, 2016.

[26] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu. Speeding up k-means algorithm by gpus. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 115–122. IEEE, 2010.

[27] Zhehao Li, Jifang Jin, and Lingli Wang. High-performance k-means implementation based on a simplified map-reduce architecture. *arXiv preprint arXiv:1610.05601*, 2016.

[28] Top500 list. https://www.top500.org/lists/2018/06/.

[29] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[30] James Newling and François Fleuret. Fast k-means with accurate bounds. In *International Conference on Machine Learning*, pages 936–944, 2016.

[31] James Newling and François Fleuret. Nested mini-batch k-means. In *Advances in Neural Information Processing Systems*, pages 1352–1360, 2016.

[32] UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets.html.

[33] Christopher J Rossbach, Yuan Yu, Jon Currey, Jean-Philippe Martin, and Dennis Fetterly. Dandelion: a compiler and runtime for heterogeneous systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 49–68. ACM, 2013.

[34] Xiao-Bo Shen, Weiwei Liu, Ivor W Tsang, Fumin Shen, and Quan-Sen Sun. Compressed k-means for large-scale clustering. In *AAAI*, pages 2527–2533, 2017.

[35] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

[36] Leonardo Torok, Panos Liatsis, Jos Viterbo, Aura Conci, et al. k-ms. *Pattern Recognition*, 66(C):392–403, 2017.

[37] Mario Zechner and Michael Granitzer. Accelerating k-means on the graphics processor via cuda. In *Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on*, pages 7–15. IEEE, 2009.