k-Means Clustering on Two-Level Memory Systems

Michael A. Bender[*]
Stony Brook University

Jonathan Berry[†]
Sandia National Laboratories

Simon D. Hammond[†]
Sandia National Laboratories

Branden Moore[†]
Sandia National Laboratories

Benjamin Moseley[‡]
Washington University

Cynthia A. Phillips[†]
Sandia National Laboratories

ABSTRACT

In recent work we quantified the anticipated performance boost when a sorting algorithm is modified to leverage user-addressable "near-memory," which we call *scratchpad*. This architectural feature is expected in the Intel Knight's Landing processors that will be used in DOE's next large-scale supercomputer.

This paper expands our analytical study of the scratchpad to consider k-means clustering, a classical data-analysis technique that is ubiquitous in the literature and in practice. We present new theoretical results using the model introduced in [13], which measures memory transfers and assumes that computations are memory-bound. Our theoretical results indicate that scratchpad-aware versions of k-means clustering can expect performance boosts for high-dimensional instances with relatively few cluster centers. These constraints may limit the practical impact of scratchpad for k-means acceleration, so we discuss their origins and practical implications. We corroborate our theory with experimental runs on a system instrumented to mimic one with scratchpad memory.

We also contribute a semi-formalization of the computational properties that are necessary and sufficient to predict a performance boost from scratchpad-aware variants of algorithms. We have observed and studied these properties in the context of sorting, and now clustering.

We conclude with some thoughts on the application of these properties to new areas. Specifically, we believe that dense linear algebra has similar properties to k-means, while sparse linear algebra and FFT computations are more similar to sorting. The sparse operations are more common in scientific computing, so we expect scratchpad to have significant impact in that area.

Keywords

Two-Level-Memory, Scratchpad, Variable Bandwidth, k-means

1. INTRODUCTION

Memory bandwidth is not increasing fast enough to accommodate the increasing parallelism on the nodes of supercomputers and commodity systems. As a result, memory is becoming ever more distant and processors are increasingly starved for memory bandwidth.

One proposed solution to be offered by vendors is *near-memory*, also called *scratchpad* [24, 29].[1] The scratchpad is physically bonded to a package containing processing elements rather remotely available via bus. This allows vendors to devote a higher percentage of memory chip I/O to data instead of control lines. The benefit is a much higher bandwidth, compared with traditional DRAM, with similar latency. The problem is that current codes will not use the new memory efficiently without programmer intervention.

Trinity [34], the latest National Nuclear Security Administration (NNSA) supercomputer, will use the Knight's Landing processor from Intel with Micron memory. This processor chip includes the scratchpad [24].

One potential use for on-package memory is to provide a local cache of DRAM pages with a hardware or runtime-assisted mechanism for migrating data. This approach would help those applications that could take advantage of the automated support and not require any modification. However, the extra latencies associated with automatically managing memory accesses are not suitable for all applications, and will inhibit the performance of some. The alternative to automated management is a user-controlled memory, which eliminates the lookup of data locations and concentrates data movement into specific, user-defined sites. We expect that the use of user-controlled memories will provide significantly higher performance where algorithms can be modified for such an architecture. Furthermore, we believe that future increases in the complexity of memory hierarchies will make user control of at least some memories in the system a requirement.

[1]The name "scratchpad" can also refer to high-speed internal memory used for temporary calculations [11, 32], a different technology from the near-memory analyzed in this paper.

[*]Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-2424 USA. Email: bender@cs.stonybrook.edu.

[†]Computing Research Center, Sandia National Laboratories, Albuquerque, NM 87185 USA. Email: {jberry, sdhammo, bjmoor, caphill}@sandia.gov.

[‡]Washington University in St. Louis, St. Louis, MO 63130 USA. Email: bmoseley@wustl.edu.

MEMSYS '15 Washington, D.C., USA

DOI: http://dx.doi.org/10.1145/2818950.2818977

As memory bandwidth becomes ever more scanty relative to processing power, it becomes increasingly profitable to exploit advances such as scratchpad in high-performance computing.

The Center of Excellence for Application Transition [29], a collaboration between NNSA Labs (Sandia, Los Alamos, and Lawrence Livermore), Intel, and Cray, has a mandate to ensure that applications can use the new architecture when it becomes available in 2015-16. As a community, however, we are only now beginning to understand the implications of scratchpad to this goal. Locality has been a key concept in supercomputing for decades, yet the advent of scratchpad is placing further emphasis on this concept. Even the most efficient HPC codes will not be prepared to exploit scratchpad despite decades of focus on locality. The problem is that using scratchpad involves an explicit data copy, the cost of which must be also amortized by adequate reuse of that data.

For an application to benefit from a scratchpad-enhanced memory architecture, good data locality is necessary, but it is not sufficient. For example, consider a program that scans a large array once. This program has great data locality. However, running this program on a memory system with a scratchpad will not help. Moving the data to the scratchpad before computation is effectively just an extra copy.

The following definitions help describe the conditions under which a scratchpad-enhanced memory can help. A computation is *memory bound* if the limiting factor in its running time is the time to feed data to the processors. Adding more compute power does not improve runtime in any significant way, but delivering data faster will. A process is *compute bound* if the limiting factor is the total processing power. A compute-bound computation might run faster if given more or faster processors, but is unlikely to benefit from faster memory.

Intuitively, a computation can be helped by a scratchpad-enhanced memory architecture if that computation has the following properties.

1. (**Memory boundedness**) In the absence of scratchpad, any variant of the computation is memory bound.

2. (**Scratchpad chunking is appropriate**) The computation can be broken into scratchpad-sized, largely independent chunks with sufficient locality to process, and then these chunks can be efficiently reassembled.

3. (**Cache chunking is insufficiently helpful**) Breaking the computation into cache-sized chunks is insufficiently helpful. This may be because the divide-and-conquer overhead for breaking into subproblems and combining sub-soluitons is expensive or simply because larger chunks help more.

4. (**Scratchpad chunk reuse**) After being copied to scratchpad, each chunk needs to be scanned sufficiently many times to amortize the cost of the copy.

Bender et al. [13] recently proposed an algorithmic memory model for the scratchpad. Their model generalizes existing sequential and parallel external-memory models [1,5] to account for high- and normal-bandwidth memory. Specifically, the model assumes two different block sizes, B and ρB (with integer parameter $\rho > 1$) to model the bandwidths of DRAM and the larger bandwidth of the scratchpad. The model then analyzes the number of block transfers, a good proxy for the running time if the computation is memory bound. That is, moving data to and from memory has to be the dominant part of the running time.

The algorithmic scratchpad model was first applied to the problem of sequential and parallel sorting algorithms [13]. New, scratchpad-aware variants were tested with Sandia National Laboratories' Structural Simulation Toolkit (SST) extension [30] simulator, which can simulate a wide range of architectural features, including scratchpad memory. (Actual scratchpad architectures have yet to be released, so one cannot evaluate performance on a deployed system.)

Despite the imminent arrival of architectures with scratchpad, we are aware of only this one application with a published study of scratchpad performance. In this paper we give scratchpad-aware algorithms, in the model of [13], for clustering, a fundamental tool of machine learning and data mining. Clustering has applications in many fields, including social network analysis, pattern recognition, and information retrieval. In a typical clustering problem one is given a set A of N d-dimensional data points and a parameter k. The goal is to partition the data points into exactly k clusters such that points in the same partition are similar, according to some metric. In this paper we give a scratchpad optimized algorithm for the Euclidean k-means clustering problem which we define more formally in Section 3.

The k-means problem is perhaps the most widely used and well studied clustering problem. It is used from areas spanning computer vision [28] to targeted advertising [18] to bioinformatics [35]. Although the k-means clustering problem is NP-Hard [4, 9], there are several theoretically good approximation algorithms [2,8,17,33]. In practice, the most widely used algorithm is Lloyd's algorithm [27], also known as the k-means method. Lloyd's algorithm is a simple greedy procedure that is fast, efficient and returns the highest quality solutions on real data [36]. In fact, Lloyd's algorithm has been identified as one of the top 10 most influential algorithms in data mining [36]. In this paper, we adapt Lloyd's algorithm to the scratchpad model.

The k-means problem is quite computationally intensive. Even heuristics like Lloyd's algorithm require $O(Ndk)$ operations per pass. Large instances can take hours of wall-clock time for some realistic settings of these parameters. Scratchpad-based k-means could potentially enable clustering of much larger datasets on future systems.

External memory algorithms for k-means [22] partition data into subsets and perform considerable work to cluster each subset before moving on to the next. The cost to reassemble partial results into a global solution is small compared to the cost of processing the individual subsets. We consider a variant of these algorithms in which traditional DRAM is analogous to disk and scratchpad is analogous to traditional DRAM.

We analyze the reduction in data block movement for scratchpad-based k-means algorithms. Our analysis finds the values of ρ, the ratio of the scratchpad bandwidth to DRAM bandwidth, that provide benefit for the k-means computation using Lloyd's algorithm. The maximum useful value of ρ depends upon the sizes of the k-means input set (k, d) and the size of cache and the scratchpad. That is, we show quantifiably when the increased bandwidth will or will not be an asymptotic benefit for memory block transfers. We

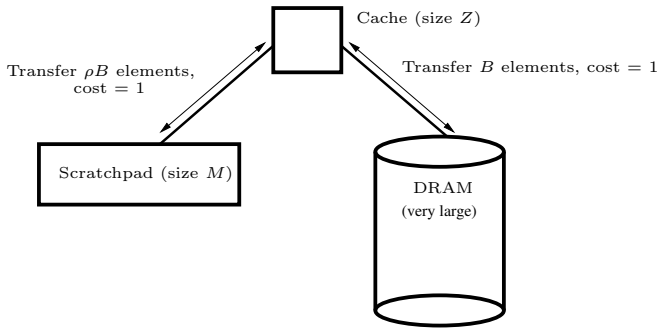

Figure 1: The scratchpad memory model [13].

give a similar analysis for a partitioning-based algorithm for k-means clustering using a scratchpad.

2. ALGORITHMIC SCRATCHPAD MODEL

This section summarizes the sequential and parallel scratchpad models of Bender et al. [13].

Scratchpad model

The scratchpad and DRAM have roughly similar access times. In contrast, the scratchpad has higher bandwidth than DRAM but smaller capacity. Both the DRAM and the scratchpad are independently connected to the cache; see Figure 1.

Data is transferred to and from the DRAM in smaller blocks of size B and to and from the scratchpad in larger blocks of size ρB, $\rho > 1$. The larger scratchpad blocks model its higher bandwidth. The cache has size Z, the scratchpad has size $M \gg Z$, and the DRAM is modeled as arbitrarily large.[2]

The cost model is simple: the performance is measured in terms of block transfers. Each block transfer costs 1, regardless of whether a large block goes to the scratchpad or a small one goes to DRAM. Computation is given for free.[3]

Observe that the scratchpad block size, ρB, in the model need not be the same as the block-transfer size in the actual hardware. But for the purpose of algorithm design, one should program assuming a block size of ρB, so that the latency contribution for a transfer is dominated by the bandwidth component.

Parallel scratchpad model

The parallel scratchpad model resembles the sequential model, except that there are p processors all occupying the same chip. Each processor has its *own* cache of size Z, and all p processors are connected to a *single* size-M scratchpad and to DRAM. In DRAM the blocks have size B and in the scratchpad ρB.

[2]Often it makes sense to assume a tall cache; that is, $Z > B^2$, or more generally, Z is at least polynomially larger than B. This is a common assumption in external-memory analysis, see e.g. [14–16, 20, 21].

[3]The scratchpad thus applies to memory-bound computations. If a computation is CPU bound, then the choice of memory architecture should matter little.

In any given *block-transfer step* of the computation, our model allows for up to p' processors to perform a block transfer, either from DRAM or from the scratchpad. Thus, although there are p processors, bandwidth limitations reduce the available parallelism to some smaller value p'. (In the sequential case, $p' = 1$, and in the fully parallel case, $p' = p$.)

Putting models in context

The scratchpad model generalizes the external-memory model of Aggarwal and Vitter [1] to include high- and low-bandwidth memory. The external-memory model assumes a two-level hierarchy comprised of a small, fast memory level and an arbitrarily large second level. Data is transferred between the two levels in blocks of size B.

The parallel scratchpad model generalizes the parallel external-memory (PEM) model [5], which has p processors each with its own cache of size M. These caches are each able to access an arbitrarily large external memory (DRAM in our terminology) in blocks of size B. These block transfers are the only way that processors communicate. This model works with CRCW, EREW, and CREW parallelism, though most research in this model (i.e. [3, 6, 31]) uses CREW.

3. K-MEANS CLUSTERING

In this section we formally describe the k-means clustering problem. We then give details of Lloyd's algorithm, the most popular k-means algorithm. Then we discuss how we can adapt this algorithm to the scratchpad architecture and bound it performance guarantees.

k-means clustering problem statement

Let A denote an input set of N points in $\mathbb{R}^d$ (d-dimensional Euclidean space). Let $v[i]$ denote the ith dimension of a point v. Let $D_{v,v'}$ denote the distance between two points v and v'.

The goal is to cluster A into k clusters, for input parameter k. Intuitively, points in the same cluster are more closely related to each other than to points in other clusters. More precisely, we can compute the variance of the pairwise distances between elements in the set of points assigned to the same cluster. This is a measure of how close the points are to each other as a group. We wish to minimize the total variance of the clustering, which is the sum of the variances over all clusters.

Say that the points in A are partitioned into k clusters $C_1, C_2, \ldots C_k$. Then, one can define the ***centroid*** of the clusters. Intuitively, the centroid is the center of mass of the cluster. In Euclidean space this is the point that is the average of all of the other points in the set. Formally, the centroid v_i of a cluster C_i has jth dimension equal to $\frac{1}{|C_i|}\sum_{u \in C_i} u[j]$. The goal of the k-means problem is to choose clusters such that $\sum_{i=1}^{k}\sum_{v \in C_i}(D_{v_i,v})^2$ is minimized.

Alternatively, one can choose a set X of k points. This naturally defines a clustering where for point $v \in X$, contains all points in $v' \in A$ such that v is the closest point in X to v'. For a set X let $D_X(v) = \min_{v' \in X} D_{v,v'}$ be the minimum distance of a point v to a point in the set X. The goal is to choose a set X of size k such that $\sum_{v \in A}(D_X(v))^2$ is minimized.

Lloyd's algorithm

We now give details of Lloyd's algorithm. We assume that $k > B$ throughout the section.

1. The algorithm is initialized with a set X of k points. This initialization can be done arbitrarily, but is usually done using a sophisticated algorithm such as k-means++ [7, 10] or by selecting the points uniformly at random [27].

2. The algorithm then assigns each point in A to the closest of the k points in X. The points in A that are assigned to v form cluster C_v.

3. The algorithm then computes the centroid of C_v and replaces v in X with the centroid of C_v. Recall that the centroid of a set of points in Euclidean space is the point that is the average of all of the other points, that is, the centroid's jth dimension is $\frac{1}{|C_v|}\sum_{u\in C_v} u[j]$.

4. The algorithm iterates until the clustering does not improve, or does not improve beyond some threshold.

Optimizing Lloyd's algorithm for cache only

We begin by analyzing Lloyd's algorithm in an architecture without a scratchpad. Later when analyzing the scratchpad-optimized algorithm, we must consider if the set X fits in the cache. Here we give a single expression of the memory accesses irrespective whether X fits in cache or not.

THEOREM 1. *An iteration of Lloyd's algorithm can be implemented in $O(\lceil kd/Z\rceil(Nd/B))$ memory accesses on a cache of size Z when the cache lines have size B.*

PROOF. The cost to scan through all N points in A once is $\Theta(Nd/B)$. Because the formula for determining the centroid is linear, if we bring the first Z/d points into cache, scan through all the points, bring the next Z/d points, into cache, and so forth, then we can compute the centroid using $O(\lceil kd/Z\rceil)$, scans through all the points. □

If $kd < Z$, then the entire set X fits in cache and the only cost is scanning through all of the points in A.

COROLLARY 1. *An iteration of Lloyd's algorithm can be implemented using $O(Nd/B)$ memory accesses, if $kd < Z$.*

Implementing Lloyd's algorithm on the scratchpad

To take advantage of the scratchpad, Lloyd's algorithm performs the following steps in each iteration:

- Recall that the algorithm's goal is to compute new centroids. For each point $v \in X$ let c_v be initialized to 0^d. This will eventually store the new centroid for the cluster assigned to v.

- The algorithm fills the scratchpad half full of points A' from A, i.e., $O(M/d)$ points.

- The algorithm fills the remaining space with a set X' of as many points of X which can fit into the other half of the scratchpad. The algorithm will iteratively bring in chunks of X until scanning through all of the points in X. For each chuck X', the algorithm determines for each point in A' its closest point X'. After scanning through all of X, the algorithm can determine which point in X' is closest to each point A'.

- Let C_v denote the points of A' which were closest to $v \in X$. The algorithm sets $c_v = c_v + \frac{1}{|C_v|}\sum_{v'\in C_v} v'$.

- The algorithm recurses on another set A' of points in A until it iterates through all points in A.

Now we bound the performance guarantees of this scratchpad optimized algorithm.

THEOREM 2. *When there is a scratchpad of size M and a cache of size Z, an iteration of Lloyd's algorithm can be implemented with $O\left(\lceil\frac{kd}{M}\rceil\lceil\frac{Nd}{M}\rceil\frac{M}{B}\right)$ DRAM accesses and $O\left(\lceil\frac{kd}{M}\rceil\lceil\frac{Nd}{M}\rceil\lceil\frac{M}{Z}\rceil\left(\frac{M}{\rho B}\right)\right)$ scratchpad accesses.*

PROOF. We first analyze the DRAM accesses. The scratchpad thus has to be loaded $O\left(\lceil\frac{kd}{M}\rceil\lceil\frac{Nd}{M}\rceil\right)$ times from DRAM and the cost for each scratchpad load is $O(M/B)$.

We next analyze the scratchpad accesses. For each of the scratchpad loads, the scratchpad runs what is effectively an iteration of Lloyd's algorithm. Using the same proof as in Theorem 1, since each iteration has $O(M/d)$ points and the block transfer size is ρB, the cost per iteration is $O\left(\lceil\frac{M}{Z}\rceil\left(\frac{M}{\rho B}\right)\right)$. □

To understand the takeaway or "moral" of Theorem 2, we separate into three cases: (1) the k points in X fit in cache ($Z \geq kd$), (2) the k points in X fit in the scratchpad but not in cache ($Z < kd \leq M$), and (3) the k points in X do not fit in the scratchpad ($M < kd$).

The following three corollaries explain the benefit of the scratchpad for each of these three cases.

We being stating that the scratchpad does not give any additional speed-up when all points in X fit into the cache.

COROLLARY 2. *If the k points in X fit in cache ($Z \geq kd$), then the scratchpad delivers no asymptotic benefit. With or without scratchpad, Lloyd's algorithm can be implemented so that the dominant memory-access cost comes from the DRAM accesses; it is $O(Nd/B)$, the cost of a linear scan.*

Now we consider the case where X is too large to fit into the cache, but fits into the scratchpad. In this case, we expect speed-up using the increased bandwidth of the scratchpad.

COROLLARY 3. *If the k points in X fit in scratchpad but not cache ($Z < kd \leq M$), then the scratchpad can accelerate Lloyd's algorithm by a factor of up to $\min\{\rho, kd/Z\}$. There is no more asymptotic benefit once $\rho \geq kd/Z$.*

PROOF. By Theorem 2, with a scratchpad there are $O(Nd/B)$ DRAM accesses and by Theorem 1, without a scratchpad, there are $O((kd/Z)(Nd/B))$ DRAM accesses.

There are $O((kd/Z)(Nd/(\rho B)))$ scratchpad accesses, which means that the scratchpad accesses are not dominant if $\rho > kd/Z$, so increasing ρ does not help asymptotically. □

Finally, we consider the case where X is too large to fit into the cache and also the scratchpad. In this case, we expect speed-up using the increased bandwidth of the scratchpad.

COROLLARY 4. *If the k points in X do not fit in the scratchpad ($kd > M$), then the scratchpad can accelerate Lloyd's algorithm by a factor of up to $\min\{\rho, M/Z\}$. There is no more asymptotic benefit once $\rho \geq M/Z$.*

PROOF. Reason: by Theorem 2, with a scratchpad there are $O(kNd^2/(MB))$ DRAM accesses and by Theorem 1, without a scratchpad, there are $O(kNd^2/(ZB))$ DRAM accesses.

There are $O(kNd^2/(\rho ZB))$ scratchpad accesses, which is not the dominant term as long as $\rho > M/Z$, so increasing ρ beyond this value does not help. □

Analyzing the algorithms with parallel processors

Now we generalize these theorems to multiple processor. Recall, that in the model considered there are p processors and up to $p' \leq p$ processors can simultaneously access the memory. Since Lloyd's algorithm linearly scans through the data, the process is fully parallelizable. In this case, the following theorem immediately follows from Theorem 1 for an architecture without a scratchpad.

THEOREM 3. *An iteration of Lloyd's algorithm can be implemented in $O(\lceil kd/Z \rceil (Nd/(Bp')))$ memory accesses on a cache of size Z when the cache lines have size B and p processors can access p' memory locations in parallel.*

From Theorem 2, we have the following.

THEOREM 4. *When there is a scratchpad of size M and a cache of size Z, and p processors can access p' memory locations in parallel, an iteration of Lloyd's algorithm can be implemented with $O\left(\lceil \frac{kd}{M} \rceil \lceil \frac{Nd}{M} \rceil \frac{M}{(p'B)}\right)$ DRAM accesses and $O\left(\lceil \frac{kd}{M} \rceil \lceil \frac{Nd}{M} \rceil \lceil \frac{M}{Z} \rceil \left(\frac{M}{p'\rho B}\right)\right)$ scratchpad accesses.*

These theorems give rise to the following corollary.

COROLLARY 5. *On a parallel process the speed-up produced by the scratchpad is the following.*

- *If $kd \leq Z$ then the scratchpad delivers no asymptotic benefit.*
- *If $Z < kd \leq M$ the scratchpad can accelerate Lloyd's algorithm by a factor of $\min\{\rho, \frac{kd}{Z}\}$.*
- *If $M \leq kd$ the scratchpad can accelerate Lloyd's algorithm by a factor of $\min\{\rho, M/Z\}$.*

4. PARTITIONING FOR K-MEANS CLUSTERING

In this section we give the asymptotic analysis for a version of the partitioning algorithm [19, 22, 23] for solving the k-means clustering problem.

Partitioning algorithm

The partitioning algorithm works as long as $kd < M$. The high level idea behind the partitioning algorithm is to reduce the size of the input by clustering portions of it and then reducing each cluster into a single representative point. Then the algorithm clusters these representative points to get a clustering of the original input. This algorithm gives an additional benefit over the previous algorithm in memory accesses when the number of iterations of Lloyd's algorithm is large.

The partitioning algorithm divides the input A into $|A|d/M$ partitions containing M/d points of size M. For each partition, the algorithm clusters them into k clusters. Each of these clusters will be reduced to a single weighted point. For each of the constructed clusters, it creates a new point located at the centroid of the cluster and weights this points with a weight equal to the number of points in the cluster. Afterwards, the algorithm then takes each of these weighted points and clusters them.

We now describe the algorithm which is adapted from [22]. When we describe the algorithm, we note that any arbitrary clustering algorithm can be used when clustering the partitions and we will denote this algorithm by $\mathcal{A}$. We will later discuss the performance guarantees when $\mathcal{A}$ is Lloyd's.

1. The partitioning algorithm arbitrarily partitions the input set of points A into Nd/M sets, $S_1, S_2, \ldots, S_{Nd/M}$ sets, each containing M/d points.
2. Let $\mathcal{A}$ denote any algorithm that clusters M/d data points inside the scratchpad. The algorithm sequentially brings each set S_i and clusters them separately using $\mathcal{A}$ to get k clusters $C_{i,1}, C_{i,2}, \ldots C_{i,k}$. For each cluster $C_{i,j}$, the algorithm computes the centroid $c_{i,j}$ and gives this centroid weight $w_{i,j} = |C_{i,j}|$.
3. At the end of the algorithm, there are $k \cdot Nd/M$ different centroids.
4. Finally the algorithm clusters the $k \cdot Nd/M$ centroids using the algorithms from Section 3 where the algorithm treats the weights as if there are $w_{i,j}$ points located at $c_{i,j}$. If there are too many centroids to fit into the scratchpad, the algorithm can recurse.

Using the analysis of [22], this algorithm has the following theoretical guarantee. We note that Lloyd's has no approximation guarantee, but, intuitively, the following theorem states that the quality of the solution should not be too different between using Lloyd's or the partitioning based algorithm.

THEOREM 5 ([22]). *This algorithm is a $O(\alpha^2)$-approximation algorithm where α is the approximation of the procedure $\mathcal{A}$.*

Let T denote the number of iterations of Lloyd's algorithm if it were run on the entire dataset. Let T' be the maximum number of iterations Lloyd's algorithm takes if it were used as the algorithm $\mathcal{A}$ in the partition based algorithm anytime it is invoked. If $\mathcal{A}$ is set to be Lloyd's algorithm, the partitioning algorithm gives rise to the following performance guarantees when run on a single processor machine.

THEOREM 6. *When $kd < M$, the total number of DRAM accesses by the partitioning algorithm where $\mathcal{A}$ is Lloyd's algorithm (including all iterations of Lloyd's) is $O\left(\lceil \frac{Nd}{M} \rceil \frac{M}{B}\right)$ and the total number of scratchpad accesses is $O\left(T' \lceil \frac{Nd}{M} \rceil \lceil \frac{kd}{Z} \rceil \left(\frac{M}{\rho B}\right)\right)$.*

PROOF. First we consider the DRAM accesses. The dominate factor is clustering the partitions in the first step.

Whenever we use Lloyd's algorithm, the set of k centers always fits in the scratchpad since we have assumed $kd < M$ and therefore they do not need to be placed in the scratchpad as in the previous section. There are $\frac{Nd}{M}$ partitions to be clustered. We only load each partition once into the scratchpad and loading each partition requires M/B accesses from DRAM. Thus, the total number of accesses is $(Nd/M)(M/B)$.

Now we consider the scratchpad memory accesses. For any partition, Lloyd's algorithm runs for at most T' iterations. Further, a single iteration of Lloyd's algorithm scans through all the points in M for each set of $\lceil kd/Z \rceil$ centroids. There are $\lceil Nd/M \rceil$ partitions that are considered, so the total number of accesses is $O\left(T' \left\lceil \frac{Nd}{M} \right\rceil \left\lceil \frac{kd}{Z} \right\rceil \left(\frac{M}{\rho B}\right)\right)$. □

We note that the previous two bounds can be scaled by p' if there are p processors that can access p' memory locations in parallel, since the algorithms can be fully parallelized as before.

The number of iterations of Lloyd's algorithm cannot be bounded. However, when the partitioning algorithm is used, we have observed that usually $T' \leq T$. Assuming this, we have the following corollary when comparing against the standard Lloyd's algorithm. This corollary shows that the speed-up scales with ρ up to $T'kd/Z$, allowing for a larger increase in speed-up by a factor of T' over just simulating Lloyd's using the scratchpad when $kd < M$. Further, in this case we achieve speed-up even in the case that $kd < Z$, which was not possible for the scratchpad implementation of Lloyd's algorithm.

COROLLARY 6. *The partitioning algorithm when $\mathcal{A}$ is set to be Lloyd's algorithm, $Z < kd < M$ and $T' \leq T$ achieves a speed up of $\min\{\rho, T'kd/Z\}$. This algorithm achieves speed up up to a factor of $\min\{\rho, T'\}$ when $T' \leq T$ and $kd < Z$.*

5. EXPERIMENTS TO CORROBORATE THE THEORY

We now describe our experiments running the partitioned version of Lloyd's algorithm described in Section 4. Recall that a major step in this algorithm is to run classical Lloyd's algorithm from the scratchpad using a scratchpad-sized chunk of the larger input data. We discuss the conditions under which Corollary 3 of Theorem 2 applies. We show that the partitioned k-means algorithm for our data sets run on our specific machine satisfies each of the four properties introduced in Section 1 along with a k-means-specific property. We describe our experimental design and results.

Our experimental platform is an 8-core NUMA node with approximately 21MB of cache, 32GB of DRAM, a standard memory bandwidth of approximately 25GB/s, and a processing capability of roughly 166 GFLOPS. This is roughly representative of the Trinity nodes that motivate this work. The system has been modified to mimic two-level memory via a different socket at 12GB/s to mimic the far memory. The programmer controls which data gets gets stored in near and far memory.

We ran experiments on the "ElectricityLoadDiagrams20112014" dataset from the Machine Learning Repository [26]. This dataset has 140,256 dimensions and 370 points. Our code was based on an open-source OpenMP code from Northwestern University [25]. We implemented the scratchpad-aware partition algorithm for k-means and used the code from Northwestern to implement the k-means algorithm for a subproblem in scratchpad. That is, the Northwestern code is Algorithm $\mathcal{A}$ in step 2 of the partition algorithm in Section 4.

5.1 Required Properties

Property 0: The k working centers do not all fit in cache.

This follows directly from Corollary 2. We call it out here because it is important for the other properties.

Property 1: Memory boundedness.

Before implementing a scratchpad-aware algorithm we must ensure that the algorithm is memory bound if run with no scratchpad available.

Once more, let N be the number of points, each with d dimensions. Let k be the number of centers. Each iteration of Lloyd's algorithm for k-means requires $O(Nkd)$ operations. In the code used, there are $3NKd$ operations. The amount of memory transfer during a Lloyd's iteration (not counting reuse) is $(N + k)d$. A simple calculation shows a regime where the computation will be memory bandwidth-bound.

$$\frac{3Nkd}{x} < \frac{(N+k)d}{y}$$

where, x is the processing rate in GFLOPS and y is the memory bandwidth in double precision values per second. Given our system parameters, we have:

$$\frac{3Nkd}{166} < \frac{(N+k)d}{1.5}.$$

We drop the $kd/1.5$ term on the right side since satisfying this inequality with a smaller righthand side makes memory-boundedness even more likely. Rearranging, we find that the N's and d's cancel, leaving us with an approximate upper bound on k, the number of centers: $k < 36$. Intuitively, we have just formalized the notion that k-means with large numbers of centers provides a large amount of computation per memory transfer, making it compute-bound.

Computing on scratchpad-resident data imposes yet a tighter constraint. Recall that ρ is the scratchpad bandwidth expansion parameter. In our case, $\rho = 2$. To remain memory-bandwidth bound during scratchpad computation (and hence to benefit from increases in ρ), we must have:

$$\frac{3Nkd}{x} < \frac{(N+k)d}{\rho y}$$

and hence $k < 18$.

Property 2: Scratchpad chunking is appropriate.

Our scratchpad-aware Lloyd's iteration is an implementation of the partitioning algorithm of [22]. This breaks the problem into scratchpad-appropriate chunks that can be processed independently. Running Lloyd's algorithm on the scratchpad-resident subproblem reduces a (large) scratchpad-sized chunk of data into a data set of size $O(kd)$. This is the size of the centers for the subproblem. These centers of the subproblem, taken over all subproblems, then form a final k-means problem that determines the solution.

Table 1: Predicted reductions in far DRAM access and observed runtime improvements

k	Speedup	Predicted Drop in Far Memory Accesses
10	1.33	1.02
14	1.34	1.42
18	1.36	2.0

Thus the "reassembly" of the subproblems into a final solution is efficient compared to the overall computation.

Property 3: Cache chunking is insufficiently helpful.

Because of Property 0 (from Corollary 2), for scratchpad to have benefit for k-means, we assume that the working set of k centers cannot fit into cache. That is, the cache size is less than kd. As described in the discussion of Property 2, in the partitioned algorithm, each subproblem is reduced to size kd. Even the output of the sub-computation cannot fit into cache. In this setting, with no scratchpad, there is no benefit to running a partitioned algorithm. It is impossible to keep the data set cache resident. One would just run regular Lloyd's algorithm. Under these circumstances, the partitioned algorithm will likely get a lower quality clustering and there is no reason to believe it will have fewer DRAM block transfers.

Property 4: Scratchpad chunk reuse.

Large k-means instances tend to require at least several passes before convergence. Thus, we expect to amortize the cost of copying a partition to scratchpad via sufficient numbers of passes through the data.

It's possible cache-friendly computation from scratchpad becomes compute bound. The added bandwidth is enough to reduce memory access time below compute-time requirements. If the computation without scratchpad is memory bound, then there is still benefit for using scratchpad, but no increased benefit with larger bandwidth ratio ρ.

From Property 0, for a Lloyd's iteration, the set of cluster centers does not fit into cache. If it did, there would be no intuitive or theoretical need for scratchpad. In our case, that means enforcing the inequality $kd > 21$MB. Given our existing upper bound $k < 18$, and the fact that our instance uses double precision values, we must have $d > 152,917$.

5.2 Experimental design

There are two fundamental problems with these constraints. The first is that our regime of expected performance boost invokes the "curse of dimensionality" [12], which consists of several properties that characterize high-dimensional Euclidean spaces and question the usability of distances in that context. However, recent work [37] suggests that this might not be a conclusive statement, and is at least an open research area. Variants of clustering the full space may hold promise, and scratchpad-aware versions of them might eventually help explore this research area.

The second and more immediate problem is that our lower bound on d exceeds the number of dimensions in our test problem, so we do not expect a benefit from scratchpad. Running the instance, we actually do see a small performance boost. However, in order to evaluate our theory, we have modified the problem by duplicating each dimension. The resulting $280,512$-dimensional problem allows the following experiments.

With our synthetically-doubled dataset, we find that cache will be flooded by roughly nine centers. Combining this with our bound $k < 18$, we can test an experimental range of $9 \leq k \leq 18$. Selecting $k = 10, 14, 18$, Corollary 3 leads us to predict relative speedups of 1.02, 1.42, and 2.0 for these experiments, assuming a direct relationship between reduced far-DRAM block transfers and runtime.

5.3 Experimental results

In our experiments, we find that the penalty in objective value is far less severe than that of the theoretical approximation bound.

Table 1 shows the results of our sweep through the center counts that both flood cache and keep the problem memory bound. We do observe increasing runtime speedups with increasing numbers of centers. However, we did not collect true DRAM access counts. Although the 30% runtime speedups do not match the predicted DRAM access reductions, we note that many runs with parameter settings outside of this regime of predicted benefit showed no runtime performance gain at all.

6. CONCLUSIONS

Through careful study of the most popular algorithm for k-means and future supercomputing resources with two-level memory, we have designed, analyzed, and evaluated a scratchpad-aware variant that achieves 30% speedup on a customized machine. We find that the regime of expected speedup is limited to high-dimensional problems with small numbers of cluster centers. The curse of dimensionality may therefore limit the practical impact of our algorithm. However, there has been recent interest in techniques for clustering points in high-dimensional space without first performing dimensionality reduction. Our work could contribute to and/or benefit from such research.

Further work is needed to explore scratchpad-aware variants of many more algorithms. We think that dense linear algebra is likely to share some properties of the k-means instances we have studied, and that sparse linear algebra and FFT will behave like the sorting problems we studied before. However, these are but conjectures that must be verified.

Although we have now thoroughly explored both sorting and k-means problems, our methodology is still developing. On-package, near-memory, or scratchpad, is imminent. However, applications are not ready to exploit it fully. Addressing this situation is an acute need for the supercomputing community.

Acknowledgments

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. Michael Bender was also supported by the following NSF grants: CNS 1408695, CCF 1439084, IIS 1247726, IIS 1251137, CCF 1217708, and CCF 1114809.

7. REFERENCES

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, Sept. 1988.

[2] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *23rd Annual Conference on Neural Information Processing Systems (NIPS)*, pages 10–18, 2009.

[3] D. Ajwani, N. Sitchinava, and N. Zeh. Geometric algorithms for private-cache chip multiprocessors. In *Proceedings of the Eighteenth Annual European Symposium on Algorithms (ESA)*, pages 75–86. 2010.

[4] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

[5] L. Arge, M. T. Goodrich, M. Nelson, and N. Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 197–206, 2008.

[6] L. Arge, M. T. Goodrich, and N. Sitchinava. Parallel external memory graph algorithms. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–11, 2010.

[7] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1027–1035, 2007.

[8] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.

[9] P. Awasthi, M. Charikar, R. Krishnaswamy, and A. K. Sinop. The hardness of approximation of euclidean k-means. *CoRR*, abs/1502.03316, 2015.

[10] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *PVLDB*, 5(7):622–633, 2012.

[11] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad Memory: A Design Alternative for Cache On-chip Memory in Embedded Systems. In *Proceedings of the Tenth International Symposium on Hardware/Software Codesign (CODES)*, pages 73–78, 2002.

[12] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[13] M. A. Bender, J. Berry, S. D. Hammond, K. S. Hemmert, S. McCauley, B. Moore, B. Moseley, C. A. Phillips, D. Resnick, and A. Rodrigues. Two-level main memory co-design: Multi-threaded algorithmic primitives, analysis, and simulation. In *Proc. 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Hyderabad, INDIA, May 2015.

[14] M. A. Bender, R. Ebrahimi, J. T. Fineman, G. Ghasemiesfeh, R. Johnson, and S. McCauley. Cache-adaptive algorithms. In *Proceedings of the Twenty-Fifth Symposium on Discrete Algorithms (SODA)*, pages 116–128, 2014.

[15] G. S. Brodal, E. D. Demaine, J. T. Fineman, J. Iacono, S. Langerman, and J. I. Munro. Cache-oblivious dynamic dictionaries with update/query tradeoffs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1448–1456, 2010.

[16] G. S. Brodal and R. Fagerberg. On the limits of cache-obliviousness. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC)*, pages 307–315, 2003.

[17] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.

[18] M. Danilevsky and E. Koh. Information graph model and application to online advertising. In *Proceedings of the 1st Workshop on User Engagement Optimization*, UEO '13, pages 11–14, 2013.

[19] A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *Proc. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 681–689, 2011.

[20] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual ACM Symposium on Foundations of Computer Science (FOCS)*, pages 285–297, 1999.

[21] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. *ACM Transactions on Algorithms*, 8(1):4, 2012.

[22] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. Knowl. Data Eng.*, 15(3):515–528, 2003.

[23] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, 2000.

[24] http://www.hpcwire.com/2014/06/24/micron-intel-reveal-memory-slice-knights-landing/.

[25] W. Liao. Parallel k-means data clustering, 2011. code.

[26] M. Lichman. UCI machine learning repository, 2013.

[27] S. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theor.*, 28(2):129–137, Sept. 2006.

[28] H. M. Moftah, W. H. Elmasry, N. El-Bendary, A. E. Hassanien, and K. Nakamatsu. Evaluating the effects of k-means clustering approach on medical images. In *12th International Conference on Intelligent Systems Design and Applications, ISDA 2012, Kochi, India, November 27-29, 2012*, pages 455–459, 2012.

[29] http://nnsa.energy.gov/mediaroom/pressreleases/trinity.

[30] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The Structural Simulation Toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38(4):37–42, Mar. 2011.

[31] N. Sitchinava and N. Zeh. A parallel buffer tree. In *Proceedings of the Twenty-Fourth ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 214–223, 2012.

[32] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel.

Assigning program and data objects to scratchpad for energy reduction. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 409–415, 2002.

[33] M. Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM J. Comput.*, 34(2):405–432, 2004.

[34] http://insidehpc.com/2014/07/cray-wins-174-million-contract-trinity-supercomputer-based-knights-landing.

[35] G. C. Tseng. Penalized and weighted k-means for clustering with scattered objects and prior information in high-throughput biological data. *Bioinformatics*, 23(17):2247–2255, Aug. 2007.

[36] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. F. M. Ng, B. Liu, P. S. Yu, Z. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2008.

[37] A. Zimek, E. Schubert, and H.-P. Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012.