Fast Parallel Initialisation for k-means
(Supplementary Material)

Anonymous Author(s)
Affiliation
Address
email

1 Fine grained analysis of D^2-seeding w.r.t. parameter N

We do a finer grained analysis of our algorithm's behaviour with respect to the parameter N. The following table gives the mean value of the cost taken over 10 runs.

N	MNIST ($\times 10^{10}$)	CIFAR ($\times 10^{10}$)	3D ($\times 10^{6}$)	Birch1 ($\times 10^{12}$)	Birch2 ($\times 10^{10}$)	Birch3 ($\times 10^{12}$)
k	31.98	82.27	17.22	140.97	72.51	50.37
$2k$	22.65	57.68	14.61	131.82	54.72	47.92
$5k$	21.42	54.81	12.90	121.09	51.53	45.60
$10k$	20.99	53.05	10.97	122.42	50.50	45.49
$15k$	21.33	54.30	11.46	120.21	49.29	45.52
$20k$	21.42	52.86	11.68	121.53	49.98	44.59
$30k$	21.11	53.84	10.87	121.08	50.39	44.95
$40k$	21.56	53.36	12.45	120.29	50.27	44.79
$50k$	21.20	53.60	11.32	121.13	49.52	45.05
$60k$	21.18	52.90	11.66	119.77	49.87	44.61
$70k$	20.99	53.08	11.31	123.27	49.17	44.90
$80k$	21.13	53.68	11.16	121.38	50.02	46.02
$90k$	21.08	52.67	11.64	121.20	50.84	45.12
$100k$	21.17	52.80	13.56	121.93	50.30	45.12

Table 1: This table gives the cost of the solution produced by our seeding algorithm as a function of the input parameter N.

Figure 1 gives a plot for the above results. It is clear from the results that the improvement in cost obtained by increase of N tends to diminish after $N = 10k$.

2 Comparing parallel seeding algorithms with parallel Lloyds with random seeding

In the main writeup, we have given the cost/time comparison of parallel versions of various seeding algorithms. Here, we add the cost/time values of the parallel version of the Lloyds algorithm with random seeding. This is to show that seeding algorithms indeed do give cost/time advantages. This is just to make sure that the quest for good "seeding" algorithms are well motivated. The pseudocode for the parallel version of Lloyds algorithm is given in Algorithm 2.1.

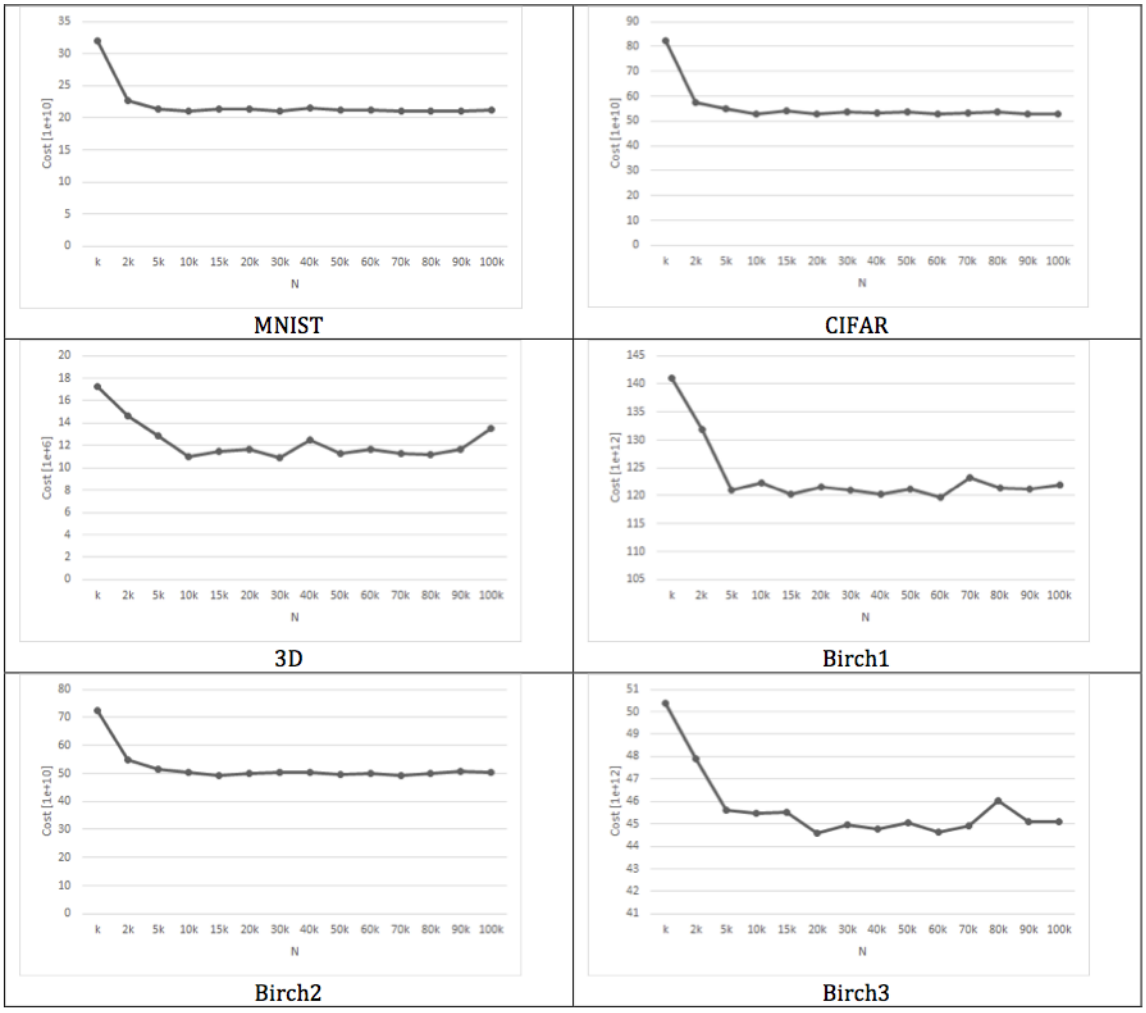

Figure 1: Plot of cost with respect to N for D^2-`seeding` algorithm.

Algorithm		MNIST		CIFAR		3D	
		Cost ($\times 10^{10}$)	Time (in sec.)	Cost ($\times 10^{10}$)	Time (in sec.)	Cost ($\times 10^{6}$)	Time (in sec.)
Parallel Lloyds with random seed	mean	31.97	9.75	78.19	32.48	40.75	4.02
	sd	1.27	4.95	6.11	9.55	23.70	0.98
Par-k-means++	mean	32.42	1.91	80.45	5.68	17.82	0.25
	sd	1.16	0.51	6.43	0.47	6.36	0.03
k-means$\|\|$ ($\ell = 2k, r = 5$)	mean	31.21	10.68	68.55	34.29	15.72	2.92
	sd	1.02	1.28	2.89	3.51	5.06	0.48
Par-D^2-Seeding ($N = 10k$)	mean	21.23	2.26	53.80	6.72	11.98	0.27
	sd	0.49	0.60	1.33	0.77	2.02	0.06

Algorithm		Birch1		Birch2		Birch3	
		Cost ($\times 10^{12}$)	Time (in sec.)	Cost ($\times 10^{10}$)	Time (in sec.)	Cost ($\times 10^{12}$)	Time (in sec.)
Parallel Lloyds with random seed	mean	296.02	21.95	788.84	9.80	168.08	28.24
	sd	25.04	6.07	233.23	4.30	37.34	8.49
Par-k-means++	mean	189.54	27.39	165.16	26.83	67.78	25.55
	sd	8.76	2.33	21.25	2.27	4.72	3.24
k-means$\|\|$ ($\ell = 2k, r = 5$)	mean	190.09	9.93	156.26	10.15	67.73	10.21
	sd	10.86	0.65	20.28	0.72	3.43	0.76
Par-D^2-Seeding ($N = 10k$)	mean	120.63	37.16	50.40	36.51	45.66	38.50
	sd	8.76	2.33	1.84	1.35	1.09	1.52

Table 2: Running time and cost for parallel algorithms. The number of machine threads used is $M = 40$.

Parallel-lloyds (X, k, C)
(1) Repeat until convergence:
(a) Forall $1 \leq j \leq k$, $P[j] \leftarrow \bar{0}$, $numPoints[j] \leftarrow 0$.
(a) **Parallel-for** $i \leftarrow 1$ to $|X|$:
(i) $j \leftarrow \arg\min_l \left\{||X[j] - C[l]||^2\right\}$.
(ii) $P[j] \leftarrow P[j] + X[i]$; $numPoints[j] \leftarrow numPoints[j] + 1$.
(b) **For** $i \leftarrow 1$ to k
(i) $C[i] \leftarrow \frac{P[i]}{numPoints[i]}$.
(2) Return C.

Algorithm 2.1: The parallel version of the Lloyds algorithm.

3 Comparing k-means++seed, k-means$||$, and D^2-seeding

All three algorithms, namely k-means++seed, k-means$||$, and D^2-seeding are sampling based algorithms with small differences. It is important to understand these subtle differences between these algorithms. k-means++seed samples k points in k sequential steps where there is a clear forward dependency. The sampling distribution of the i^{th} step depends on the centers chosen in the previous $(i-1)$ steps. This forward dependency restricts parallelization since there is a k size sequential component. The k-means$||$ algorithm was designed precisely to avoid this sequential component. The main observation made was that if in each step, we sample more than one point independently then the sampled centers over even smaller number of iterations behave well. So, if we sample roughly $\ell = 2k$ points independently in each iteration over as small as $r = 5$ iterations, then the sampled points are nice with respect to the cost. However, the benefits (in terms of cost) of sampling points in a sequence as in k-means++seed is compromised a bit. On the other hand, since the sequential component has been reduced, the algorithm allows faster parallel versions. This is the reason why the running time of k-means$||$ is consistently better than the other algorithms

when k is large (for Birch datasets k is 100). The D^2−seeding algorithm, takes a slightly different approach with the main goal of obtaining solutions with smaller cost. It again picks centers in k sequential steps as in the k−means++seed algorithm. However, it tries to pick a "better" center in each step. It does so by independently sampling lots of points in each step. It then clusters these independently sampled points into k clusters, picks the largest cluster, and then takes the centroid of the largest cluster.

The following hypothetical dataset highlights the difference between these three algorithms. We note that on this hypothetical 1-dimensional dataset, the k−means++seed and D^2−seeding algorithms give the optimal result whereas the k−means||algorithm gives an arbitrarily bad result. Consider a large k (say $k = 100$). There are 2^k points co-located at origin, $10k$ point co-located at 1, $10k$ points co-located at k, $10k$ points co-located at k^2, ..., $10k$ points co-located at k^{k-1}. Clearly, the optimal k-means cost for this dataset is 0. Now, both the k−means++seed and D^2−seeding algorithms on this dataset will pick one center each from the k different locations and hence give the optimal solution. However, note the behaviour of k−means|| with $\ell = 2k$ and $r = 5$. The first center is chosen uniformly at random, so most likely this will be a point from the origin. Subsequently, it will pick roughly $2k$ points independently in the first iteration. Given the distribution of the points, most likely all the points will be picked from location k^{k-1}. In the second iteration, most likely all the points will be chosen from the location k^{k-2} and so on. So, at the end of $r << k$ iterations, points from a only few locations (out of k) have been sampled and due to this, an arbitrarily bad non-optimal solution is produced.