Concurrency Opportunity Recognition

Carnegie Mellon University

18-645

Jike Chong

Ian Lane

Waitlist Status

- In Total 230 students have signed up for the course: 142 slots
- Slots filled in a first come / first filled basis. Talk to your Academic Advisor (i.e. Brittany Reyes from ECE)

- Current Status
  - TA (PITTSBURGH): **126** **45** **81**
  - TS (SILICON VALLEY): **59** **52** **7**
  - TZ (GUANGZHOU, CHINA): **45** **45** **0**

- **Course Drop and Waiting List will be finalized today (Jan 30th)**
- Teams for Homework 1 will be finalized based on students officially in the course roster at the end of today
  - Due date shifted from February 1st → February 5th

Homework 1 – Feb 1st → Feb 5th

- **Course Drop and Waiting List will be finalized today (Jan 30th)**
- Teams for Homework 1 will be finalized based on students officially in the course roster at the end of today
- TeamIDs will be send out today once final course roster has been obtained
- Due date shifted from February 1st → February 5th

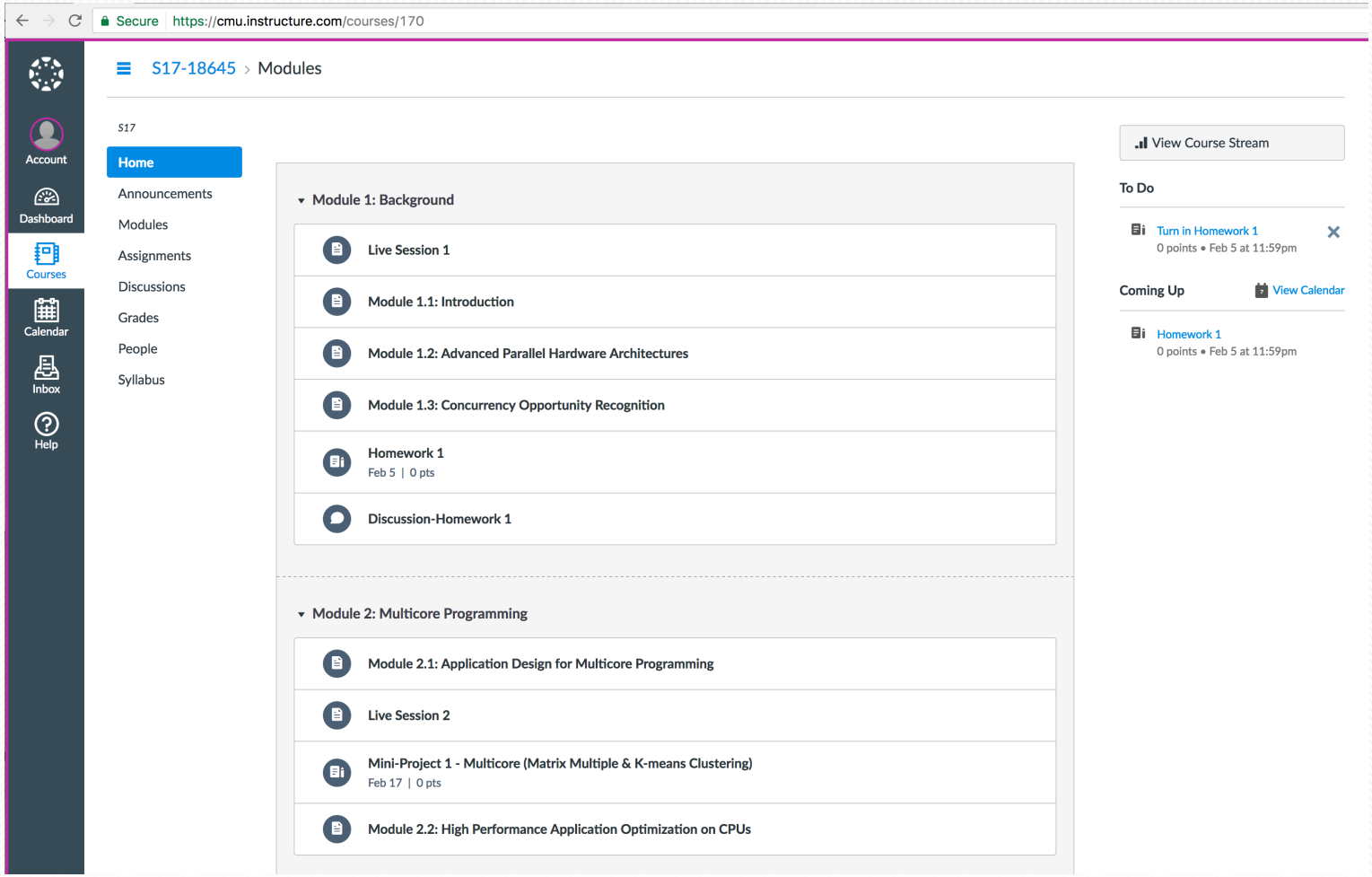
Modules Available Online

- 1.1 Introduction
- 1.2 Advanced Parallel Hardware Architectures
- 1.3 Concurrency Opportunity Recognition

- 2.1 Application Design for Multicore Programming
- 2.2 High Performance Application Optimization on CPUs
- 2.3 Performance Analysis: Roofline Model

- Homework 1 — Must Complete by February 5th
- Mini-Project 1 — Due February 17th

https://cmu.instructure.com/courses/170



Questions you should ask yourself…

- What is the difference between concurrency and parallelism?
- What are the four key elements of the human problem solving process?
- What are the characteristics of a current algorithm implementation?
- What levels of concurrency can be **exposed** in the k-mean algorithm?
- What levels of parallelism are available to be **exploited**?
- What mapping between concurrency and parallelism can be **explored**?
- How is this relevant to writing fast code?

Distinction: Concurrency vs. Parallelism

Concurrency	Parallelism
The property of **an application** that… …allows for tasks to **have the potential** to be… …**executed simultaneously**	The property of **a platform** that… …allows for tasks to **have the potential** to be… …**executed simultaneously**
The **application architecture** in which… …**more than one task** is active and able to… …**make progress** at one time	The **platform architecture** in which.. …**more than one task** can be active and… …**make progress** at same time
We **expose concurrency** in our applications.	We **exploit parallelism** in our platforms.

Problem solving for fast code

- Writing fast code is a process coherent with

“general problem solving behavior”

- Newell and Simon, Human Problem Solving (1972), pp. 72-73

- The process of problem solving involves:
  1. Understand the **current state**
  2. Observe the **internal representation**
  3. **Search** among alternatives
  4. Select from a set of **choices**

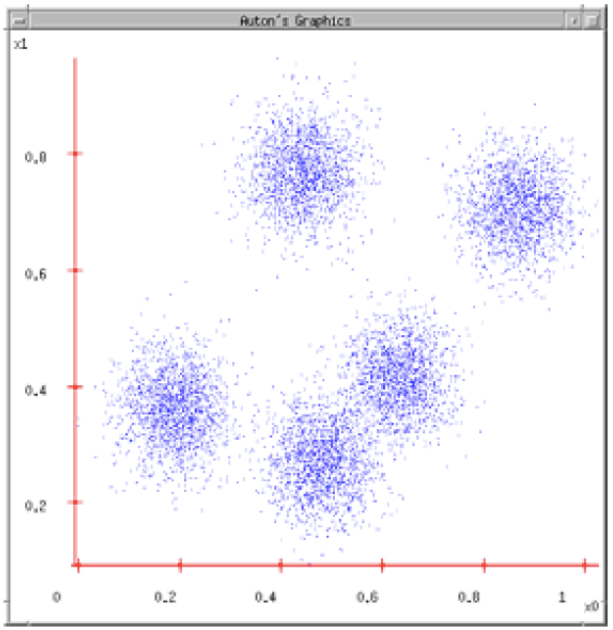
The *k*-means Problem

- Find *k* cluster centers that minimize the distance from each data point to a cluster center

- Important algorithm in machine learning:
  - Statistical data analysis
  - Vector quantization (Speech Recognition)

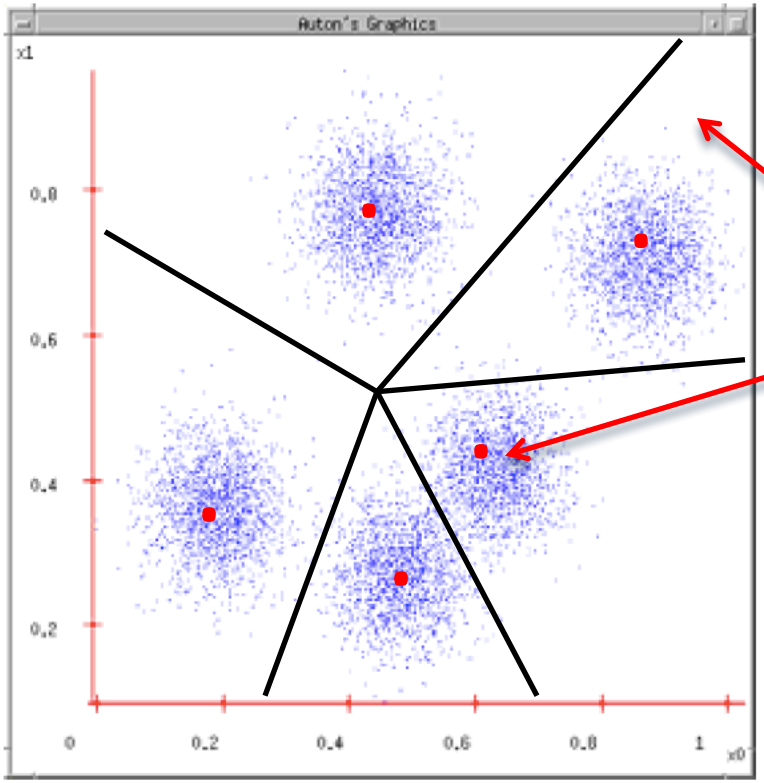
- NP-hard for arbitrary input
- ***k*-means algorithm** frequently finds a reasonable solutions quickly

- Issues:
  - Worst case running time is super-polynomial
  - Approximation can be arbitrarily bad



The *k*-means Problem

- Find *k* cluster centers that minimize the distance from each data point to a cluster center



Cluster

Cluster center (centroid)

k: Number of clusters (defined a-priori)

Cluster: Assignment of data points to a class

Cluster Center: μ of data points in a cluster

Related Problems and Algorithms

- ***k*-means++**: Maximize scattering on initial cluster centers
- **KD-trees**: Fast *k*-means - Pre-compute distance between data points
- ***x*-means**: *k*-means with efficient estimation of the number of classes

- Gaussian Mixture Models:
  - Probabilistic assignments to clusters
  - Multivariate Gaussian distributions instead of means

- Expectation-maximization algorithms (EM algorithms)
  - Find maximum likelihood estimates of parameters in a statistical model, where the model depends on unobserved latent variables.

- Expectation-maximization algorithms for Conditional Likelihoods
  - Estimate parameters in a statistical model to optimize conditional likelihood (where the objective function is a rational function)

k-means Algorithm (“*Lloyd’s algorithm*”)

- Given an initial set of *k* means $\mathbf{m}_1^{(1)},\ldots,\mathbf{m}_k^{(1)}$
- **Expectation Step:** Assign each observation to the cluster with the closest mean

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - \mathbf{m}_i^{(t)} \right\| \leq \left\| \mathbf{x}_j - \mathbf{m}_{i^*}^{(t)} \right\| \text{ for all } i^* = 1, \ldots, k \right\}$$

- **Maximization Step:** Calculate the new means to be the centroid of the observations in the cluster.

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

- Iterate until convergence or stopping criteria met

The Algorithm

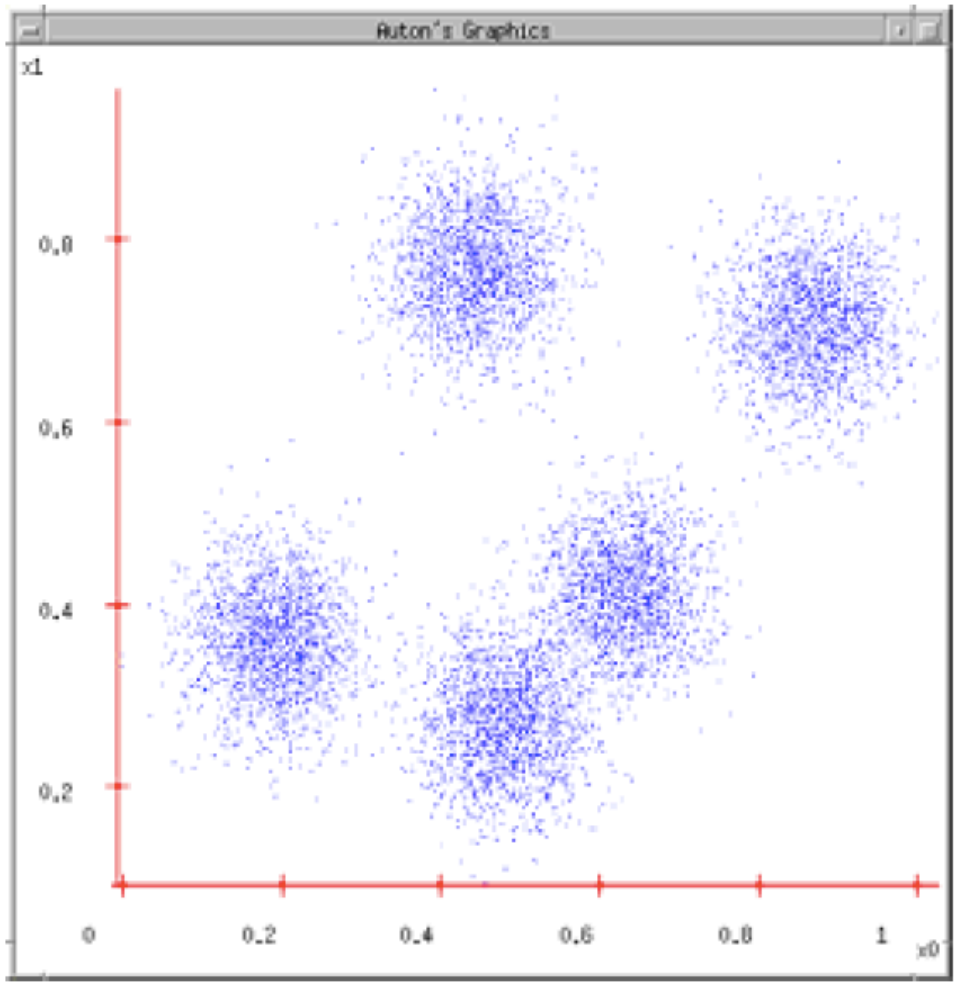
Example:

k=5

Distance metric=euclidean

Dimensions=2

1. Randomly select k cluster Centers
2. Assign closest Center to each data point
3. Update Centers based on assignments from (2)
4. Re-iterate steps 2-3 until convergence or stopping criteria met



The Algorithm

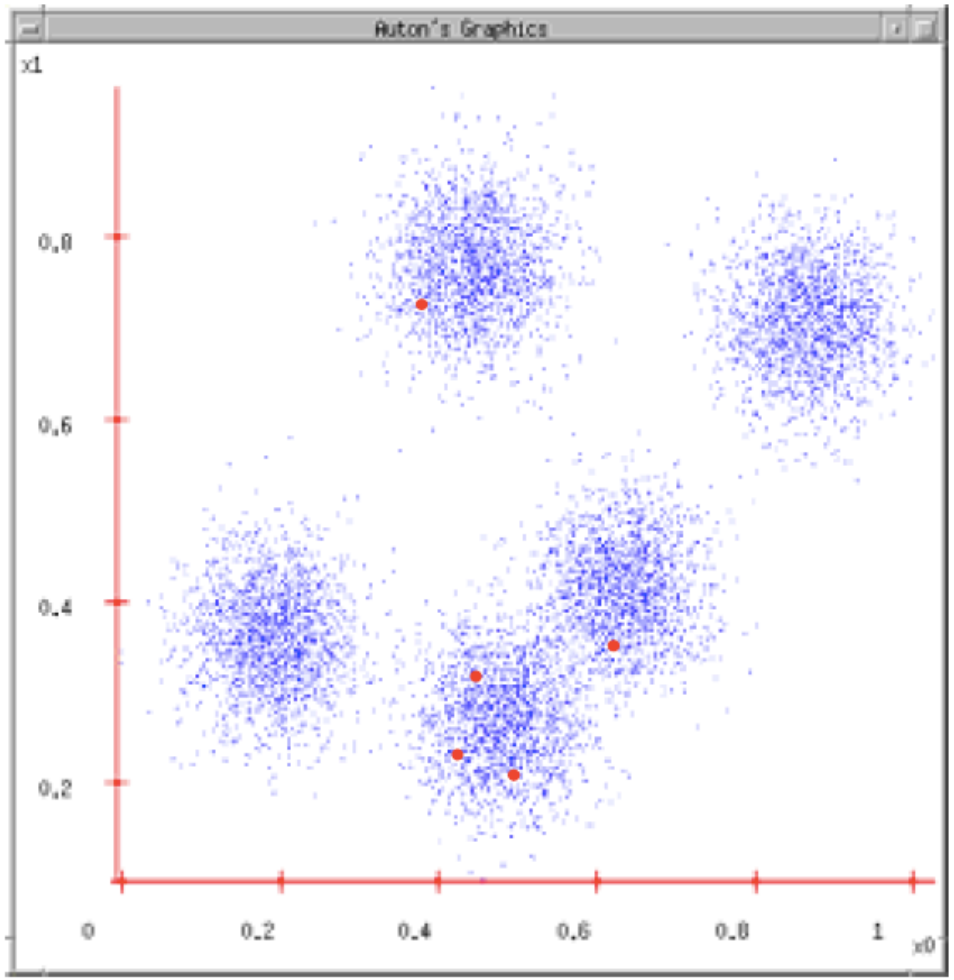
Example:

k=5

Distance metric=euclidean

Dimensions=2

1. Randomly select k cluster Centers
2. Assign closest Center to each data point
3. Update Centers based on assignments from (2)
4. Re-iterate steps 2-3 until convergence or stopping criteria met



The Algorithm

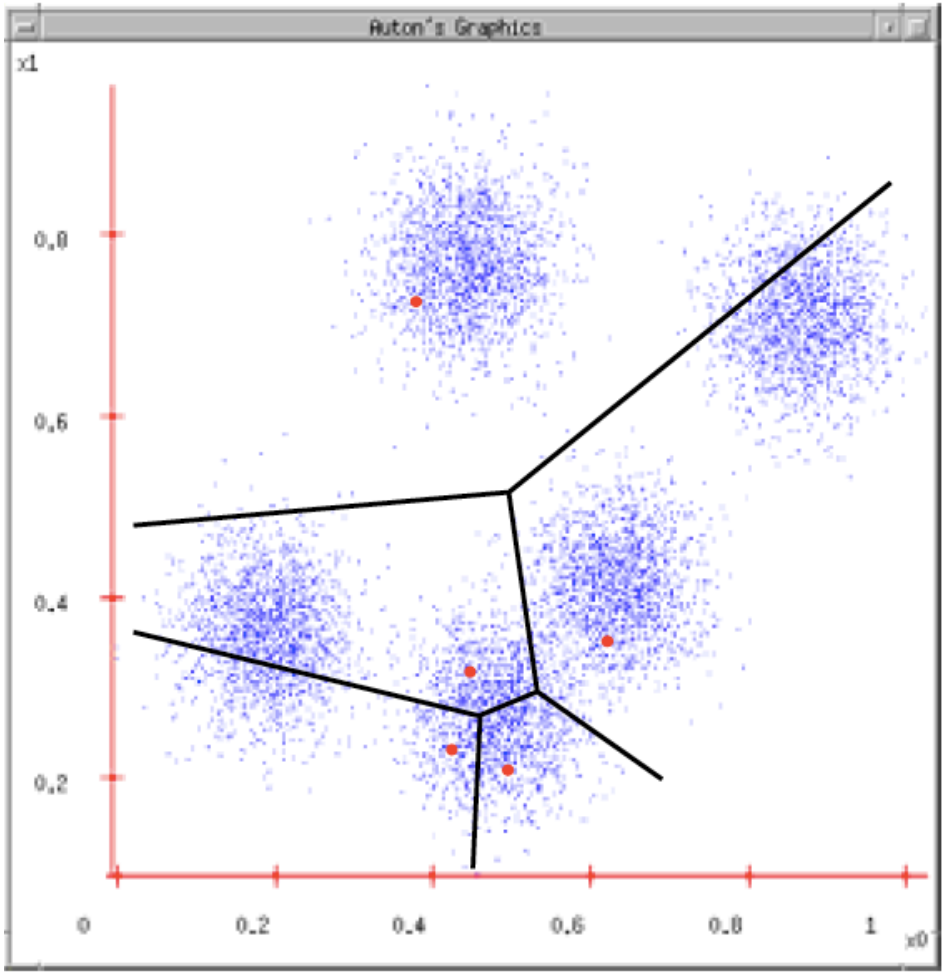
Example:

k=5

Distance metric=euclidean

Dimensions=2

1. Randomly select k cluster Centers
2. Assign each data point to closest Center
3. Update Centers based on assignments from (2)
4. Re-iterate steps 2-3 until convergence or stopping criteria met



The Algorithm

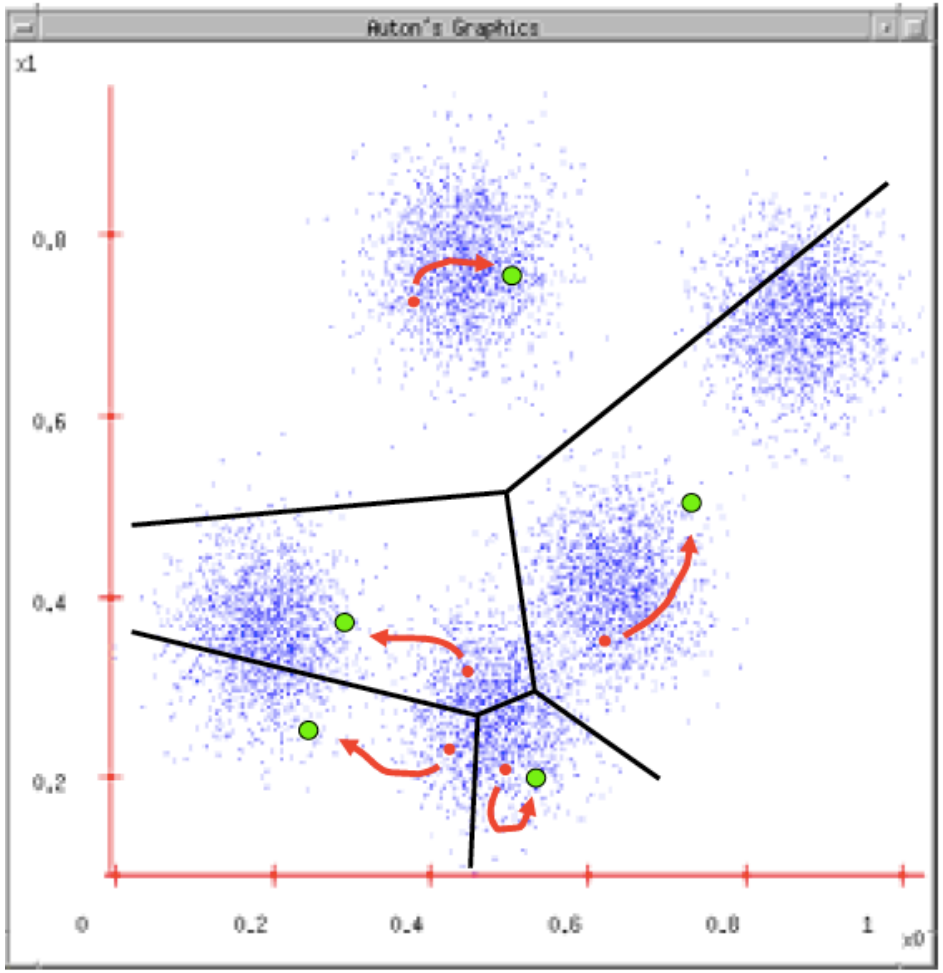
Example:

k=5

Distance metric=euclidean

Dimensions=2

1. Randomly select k cluster Centers
2. Assign closest Center to each data point
3. Update Centers based on assignments from (2)
4. Re-iterate steps 2-3 until convergence or stopping criteria met



The Algorithm

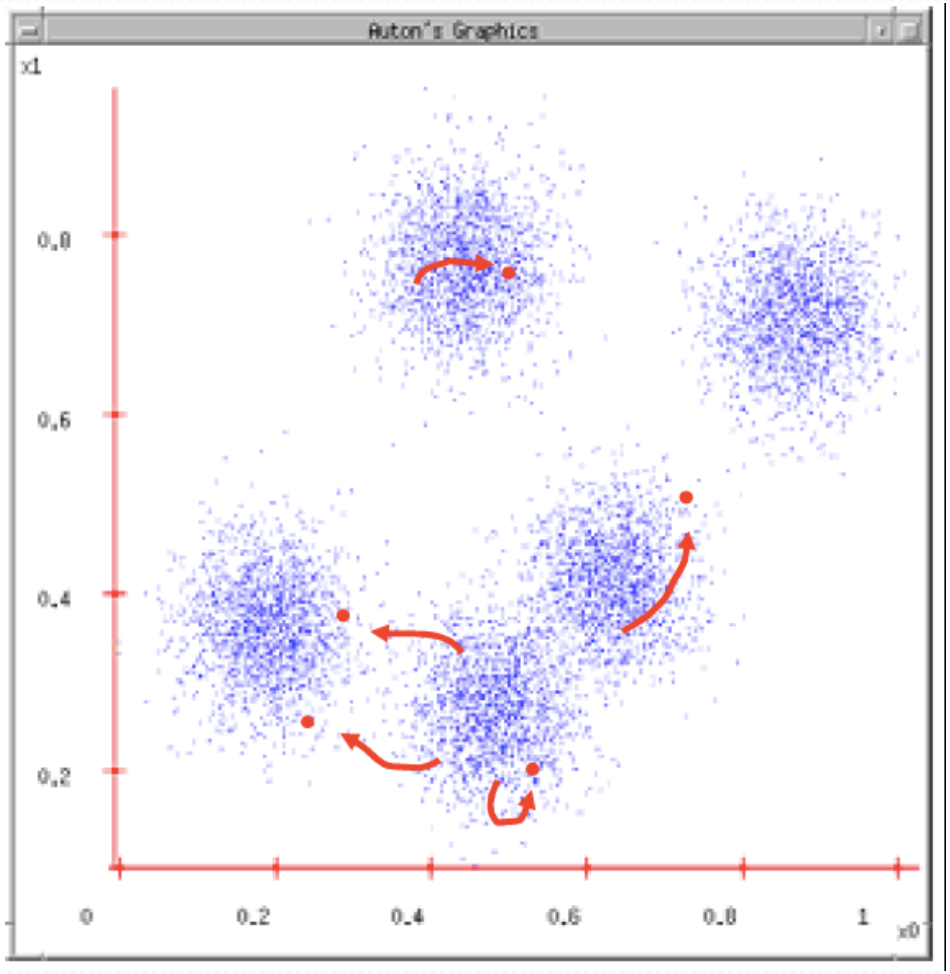
Example:

k=5

Distance metric=euclidean

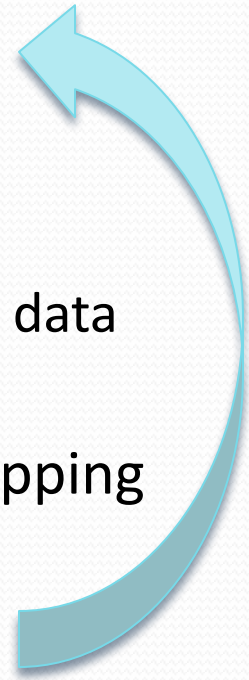
Dimensions=2

1. Randomly select k cluster Centers
2. Assign closest Center to each data point
3. Update Centers based on assignments from (2)
4. Re-iterate steps 2-3 until convergence or stopping criteria met



The Phases

1. **Initialization:** Randomly select *k* cluster centers
   - Select *k* samples from data as initial centers [Forgy Partition]
2. **Expectation:** Assign each data point go closest center
   - Compare each data point (*N*) to each cluster center (*k*)
   - Distance Metric: Euclidean distance (*D* dimensions)
3. **Maximization:** Update centers based on assignments
   - For each cluster (*k*) compute mean (D dimensions) from data points assigned to that cluster
4. **Evaluate:** Re-iterate steps 2-3 until convergence or stopping criteria met
   - Percentage of data points re-assigned
   - Number of iterations (2-3)



A Fast Implementation of *k*-means

- Writing fast code is a process coherent with

"general problem solving behavior"

- Newell and Simon, Human Problem Solving (1972), pp. 72-73

- The process of problem solving involves:
  1. Understand the **current state**
  2. Observe the **internal representation**
  3. **Search** among alternatives
  4. Select from a set of **choices**

A Fast Implementation of *k*-means

- Following the process of problem solving with *k*-means:
  1. Understand the **current state**
     - Running on a platform
     - Using a specific set of resources
     - Achieving a specific performance
     - Meeting a specific criteria/requirement
  2. Observe the **internal representation**
  3. **Search** among alternatives
  4. Select from a set of **choices**

Assumption:
Starting from a functionally correct reference implementation

Implication:
Must observe the *current state* and *implementation requirements* before starting to solve a problem

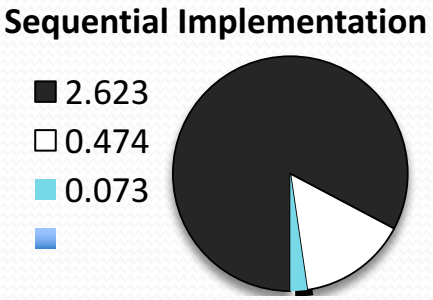
Understanding the Current State

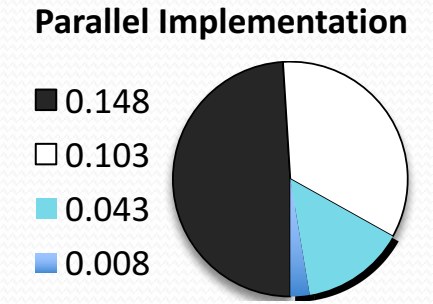
- **Running on a platform**
  - *Platform:* *Linux + GCC on x86 multicore processor*
- **Using a specific set of resources**
  - *Computation:* *2 to 8 cores, 2 to 8 way SIMD*
  - *Data:* *32KB L1, 256KB L2, shared L3 cache, 2 to 16GB DRAM*
  - *Synchronization:* *on-chip shared-memory abstraction*
- **Achieving a specific performance**
  - *As measured in Mini-Project 1*
  - *k-means:* *1.5x performance on largest set*

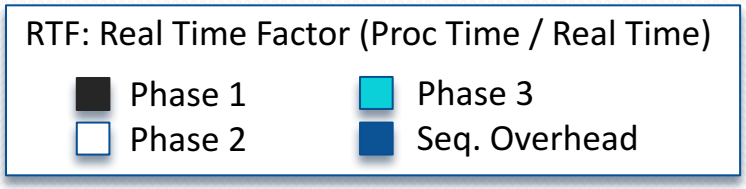
What to Measure for Performance?

- Performance Analysis: **Roofline Model**
  - Will be introduced in a later module

- Before that, a few simple techniques:
  - Observe the phases of execution
  - Characterize the execution time break downs
  - Reason about why a piece of code is slow
  - Identify performance bottlenecks







Kisun You, Jike Chong, Youngmin Yi, Ekaterina Gonina, Christopher Hughes, Yen-Kuang Chen, Wonyong Sung, Kurt Keutzer, "Parallel Scalability in Speech Recognition: Inference engine in large vocabulary continuous speech recognition", IEEE Signal Processing Magazine, vol. 26, no. 6, pp. 124-135, November 2009.

Current state: *k*-means algorithm

- 4 Phases (Initialization, Expectation, Maximization, Evaluate)
  - Majority of time spent on Expectation and Maximization phases
- Entire data set can fit in memory on a single machine
- Number of samples (*N*) and feature dimensions (*D*) vary significantly
- Evaluation for any number of clusters ($2 \geq k \leq 300$)
- Example Data Sets:
  - *ionosphere_scale*: 351 Samples, 34 Dimensions
  - *svmguide*: 7089 Samples, 4 Dimensions
  - *cod-rna*: 59535 Samples, 8 Dimensions
  - *ljcnn1*: 191681 Samples, 22 Dimensions

A Fast Implementation of *k*-means

- Following the process of problem solving with *k*-means:
  1. Understand the **current state**
  2. Observe the **internal representation**
     - Application structure
       - Identified four phases of execution
     - Implementation concerns
       - Task considerations
       - Data representations
       - Concurrency opportunities
  3. **Search** among alternatives
  4. Select from a set of **choices**

Example Code (Initialize)

kmeans/seq_kmeans.c
lines: 116-119

```
....
    /* pick first numClusters elements of objects[] as initial cluster centers*/
    for (i=0; i<numClusters; i++)
        for (j=0; j<numCoords; j++)
            clusters[i][j] = objects[i][j];
....
```

```
__inline static float euclid_dist_2(int numdims, float *coord1, float *coord2)
{
  int i;
  float ans=0.0;
  for (i=0; i<numdims; i++)
    ans += (coord1[i]-coord2[i]) * (coord1[i]-coord2[i]);
  return(ans);
}
```

Example Code (Initialize)

kmeans/seq_kmeans.c
lines: 116-119

....

Active Data Structures

objects: N x D

clusters: k x D

...

```
__inline static float euclid_dist_2(int numdims, float *coord1, float *coord2)
{
    int i;
    float ans=0.0;
    for (i=0; i<numdims; i++)
        ans += (coord1[i]-coord2[i]) * (coord1[i]-coord2[i]);
    return(ans);
}
```

Example Code (Initialize)

kmeans/seq_kmeans.c
lines: 116-119

```
....
    /* pick first numClusters elements of objects[] as initial cluster centers*/
    for (i=0; i<numClusters; i++)
        for (j=0; j<numCoords; j++)
            clusters[i][j] = objects[i][j];
....
```

```
__inline static float euclid_dist_2(int numdims, float *coord1, float *coord2)
{
  int i;
  float ans=0.0;
  for (i=0; i<numdims; i++)
    ans += (coord1[i]-coord2[i]) * (coord1[i]-coord2[i]);
  return(ans);
}
```

Define distance metric (Euclidian)

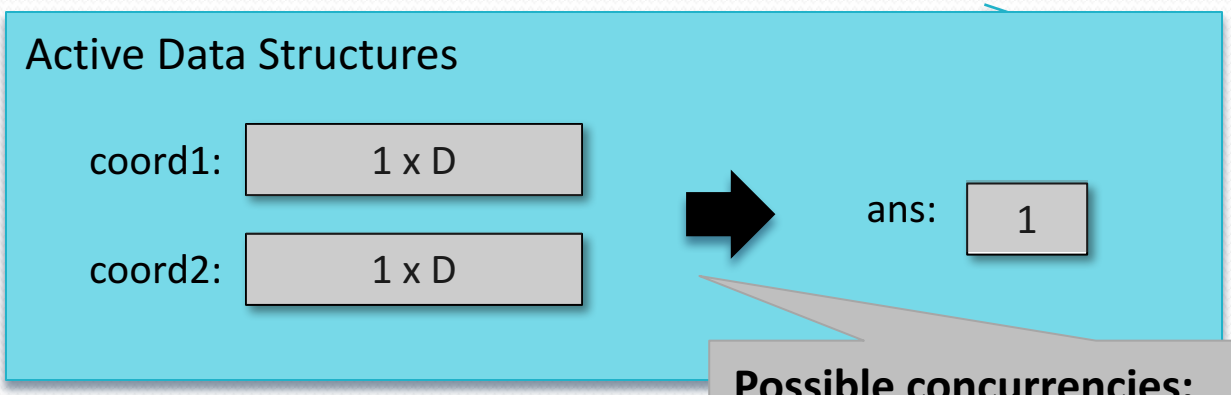
kmeans/seq_kmeans.c
lines: 51-63

Example Code (Initialize)

kmeans/seq_kmeans.c
lines: 116-119

```
....
    /* pick first numClusters elements of objects[] as initial cluster centers*/
    for (i=0; i<numClusters; i++)
        for (j=0; j<numCoords; j++)
            clusters[i][j] = objects[i][j];
....

__inline static float euclid_dist_2(int numdims, float *coord1, float *coord2)
{
}
```



Define distance metric (Euclidian)

kmeans/seq_kmeans.c
lines: 51-63

Possible concurrencies:
D (sum reduction) ← euclid_dist_2()

Example Code (Expectation)

kmeans/seq_kmeans.c
lines: 136-147

```
....
    delta = 0.0;
    for (i=0; i<numObjs; i++) {
      /* find the array index of nestest cluster center */
      index = find_nearest_cluster(numClusters, numCoords, objects[i], clusters);

            …

      /* if membership changes, increase delta by 1 */
      if (membership[i] != index) delta += 1.0;

      /* assign the membership to object i */
      membership[i] = index;
…
```

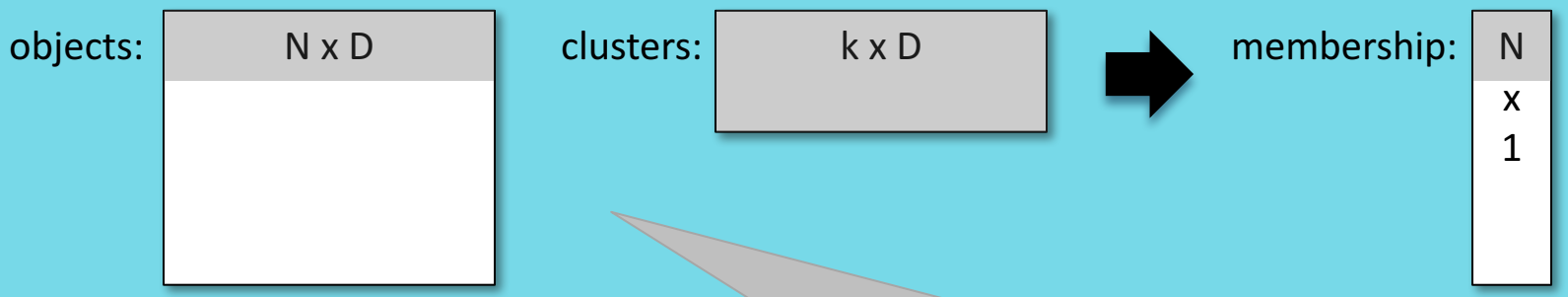
Evaluate distance to each cluster centroid and select closest

Example Code (Expectation)

kmeans/seq_kmeans.c
lines: 136-147

```
....
    delta = 0.0;
    for (i=0; i<numObjs; i++) {
      /* find the array index of nestest cluster center */
      index = find_nearest_cluster(numClusters, numCoords, objects[i], clusters);
```

Active Data Structures




```
      membership[i] = index;
...
```

Possible Concurrencies:

N (independent)
D (sum reduction) ← euclid_dist_2()
k (min reduction)

Example Code (Maximization)

kmeans/seq_kmeans.c
lines: 148-162

```
....
    /* update new cluster centers : sum of objects located within */
        newClusterSize[index]++;
        for (j=0; j<numCoords; j++)
            newClusters[index][j] += objects[i][j];
    }

    /* average the sum and replace old cluster centers with newClusters */
    for (i=0; i<numClusters; i++) {
        for (j=0; j<numCoords; j++) {
            if (newClusterSize[i] > 0)
                clusters[i][j] = newClusters[i][j] / newClusterSize[i];
            newClusters[i][j] = 0.0;   /* set back to 0 */
        }
        newClusterSize[i] = 0;       /* set back to 0 */
    }
...
```

prepare for next iteration

Example Code (Maximization)

kmeans/seq_kmeans.c
lines: 148-162

```
....
    /* update new cluster centers : sum of objects located within */
        newClusterSize[index]++;
        for (j=0; j<numCoords; j++)
            newClusters[index][j] += objects[i][j];
```

Active Data Structures

objects: N x D

membership: N x 1

NewClusters: k x D

```
        }
        newClusterSize[i] = 0;      /* set bac
    }
...
```

Possible Concurrencies:
- D (independent)
- N (Histogram computation into k bins)

Example Code (Evaluate)

kmeans/seq_kmeans.c
lines: 164-165

```
....
   delta /= numObjs;
   } while (delta > threshold && loop++ < 500);
...
```

Two stopping criteria defined

- Two stopping criteria:
  - Percentage of data points that change class < *threshold*
  - Max 500 iterations (Assign and Update)

The phases - concurrency

1. **Initialization:** Randomly select *k* cluster centers

2. **Expectation:** Assign closest center to each data point
   - *N* (independent)
   - *k* (min reduction)
   - *D* (sum reduction)

3. **Maximization:** Update centers based on assignments
   - *D* (independent)
   - *N* (Histogram computation into k bins)

4. **Evaluate:** Re-iterate steps 2-3 until convergence

Mapping Concurrency to Parallelism

- Following the process of problem solving with *k*-means:
  1. Understand the **current state**
  2. Observe the **internal representation**
  3. **Search** among alternatives
  4. Select from a set of **choices**

Search Among Alternatives

- Given the observed internal representations and the concurrency opportunities…
- What are the implementation **alternatives**?
  - Different mapping of **application concurrency** to **platform parallelism**
- The **search** process
  - More complex than one **application concurrency** to one **platform parallelism**
  - One level of concurrency could map to multiple levels of parallelism
  - May want to sequentialize some operations:
    - Some parallel operations are as **"work-efficient"** as sequential operations
    - Reduction – sequential: O(N), Parallel: O(N logN)

Mapping Concurrency to Parallelism

- How does it map to the platform?
  - SIMD level parallelism
  - Core level parallelism
- How does it map to the cache hierarch?
  - What data is required for each concurrent operation?
  - What are the synchronization points in the algorithm?
- Expectation & Maximization Phases
  - SIMD & core-level parallelism across data-points (N)
    - Update membership for each data point sequentially
      - Compute distance to each cluster center and select index with min. distance
    - Histogram computation (summation / assignment count for new clusters)
  - Other possible concurrency mappings?

A Fast Implementation of *k*-means

- Following the process of problem solving with *k*-means:
  1. Understand the **current state**
  2. Observe the **internal representation**
  3. **Search** among alternatives
  4. Select from a set of **choices**
     - Does solution met required criteria
     - How to evaluate a mapping?
       - Efficiency: Runs quickly, makes good use of computational resources
       - Simplicity: Easy to understand code is easier to develop, debug, verify and modify
       - Portability: Should run on widest range of parallel computers
       - Scalability: Should be effective on a wide range of processing elements
     - Other considerations: Practicality, Hardware, Engineering cost

Evaluate Choice

- Expectation & Maximization Phases
  - SIMD & core-level parallelism across data-points (N)
    - Update membership for each data point sequentially
    - Histogram computation (summation / assignment count for new clusters)

  → OpenMP

- How we can evaluate the choice and make a decision
  - Efficiency
  - Simplicity / Maintainability
  - Portability
  - Scalability