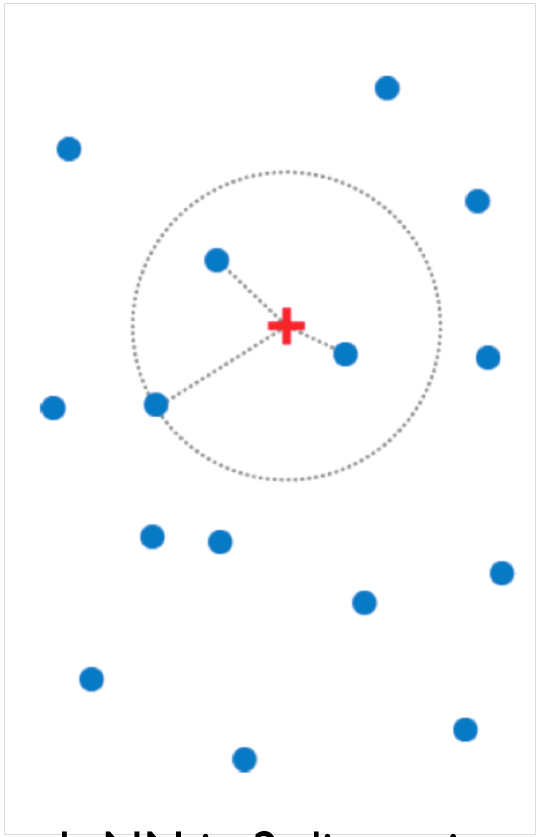
Concurrent Implementation of k-NN for WLAN Positioning

Eros Gulo
Department of Earth and Space Science, York University
November 10th, 2015

creative passionate rational confident ingenious

K–NEAREST NEIGHBOURS

- Search a set of reference points for the k-nearest points to a query point and repeat for successive query points.
- Nearness or proximity can be arbitrarily defined, though usually a metric of distance (e.g. Euclidean distance) is used.
- The query point is classified or assigned a value by its nearest neighbours.


k-NN in 2 dimensions (k = 3)

K-NN IN WLAN POSITIONING

WLAN Signal Fingerprint Matching (WSFM)

- Reference set is a database or map of the signal strength of all WLAN access points (APs) at all locations in the positioning area.
- Query point is a sample of the signal strengths of all APs (fingerprint) at user's particular location.
- k-NN used to match the fingerprint to the most similar locations (based on AP signal strength) in the database.

K-NN IN WSFM

Reference Fingerprint Set: $L = \{\ell_1, \ell_2, \ldots, \ell_m\} \quad \mathbb{R}^d$

Query Fingerprint Set: $F = \{f_1, f_2, \ldots, f_n\} \quad \mathbb{R}^d$

Distance computed for each possible pairing: $D(f_i, \ell_j) \quad j \in [1, m] \quad i \in [1, n]$

Number of APs: d

WLAN SIGNAL DIFFERENCE FINGERPRINT MATCHING (WSDFM)

Different definition of the fingerprint, namely signal strength *differences*, as opposed to absolute signal strengths.

Greatly increased dimensions of distance calculations, differences between every possible pairing of visible APs results in distance calculations in $d(d-1)/2$ dimensions, where d is the number of visible APs.

SEQUENTIAL IMPLEMENTATION

1. Initialize an array of size k to hold the k smallest distances.
2. Perform the following consecutively for each reference fingerprint:
   i. Calculate distance between reference fingerprint and query fingerprint.
   ii. If the distance is smaller than any of the distances in the distance array, place this distance in the array and remove the largest distance previously contained in the array.
3. Once all distances have been calculated, the array that remains will have the k smallest distances, and their respective reference fingerprints can be returned to complete the localization of the query fingerprint.

LASSONDE SCHOOL OF ENGINEERING | YORK UNIVERSITÉ UNIVERSITY

PROPOSED CONCURRENT K-NN ALGORITHM

The distance calculations between different f_i and ℓ_j pairings are all independent of each other, therefore, computation time can be reduced if they are computed in parallel.

Sorting of all the distances for each f_i does not have to be fully completed, therefore, computation time can also be reduced by only sorting some of the distances. The distances are only sorted until the first k-items in the list are the k-smallest and they are in order.

INITIAL CONCURRENT IMPLEMENTATION

1. Allocate arrays to hold all computed distances and their respective indices.

2. Initialize a CyclicBarrier to pause the main thread until distance calculations are complete.

3. Create individual threads to calculate each distance.

4. Each distance calculation thread calls await() on the CyclicBarrier after distance is calculated and written to the distance array.

5. Main thread calls await() prior to sorting the distance array, the CyclicBarrier allows it to proceed once all distances have been computed.

6. Main thread sorts the distances (while keeping track of their reference location indices), all distance calculation threads terminate.

INITIAL CONCURRENT IMPLEMENTATION CODE

```java
public static ArrayList<LocationCell> nearestNeighboursDif(Fingerprint print, int k)
{
    CyclicBarrier checkPoint;
    distanceArray = new ArrayList<Double>(locationDatabase.size());
    indexArray = new ArrayList<Integer>(locationDatabase.size());
    checkPoint = new CyclicBarrier(locationDatabase.size()+1);
    for (int i = 0; i < locationDatabase.size(); i++)
    {
        indexArray.add(i,i);
        distanceArray.add(i,0.0);
        (new DistanceCalculationDif(print,i,checkPoint)).start();
    }
    try { checkPoint.await(); }
    catch (InterruptedException | BrokenBarrierException e) { e.printStackTrace(); }

    // sort distance array...

}
```

INITIAL CONCURRENT IMPLEMENTATION CODE

```
private static class DistanceCalculationDif extends Thread //implements Runnable
{
    Fingerprint fP; int sID; CyclicBarrier cP;

    public DistanceCalculationDif(Fingerprint print, int spaceID, CyclicBarrier checkPoint)
    {
        fP = print;
        sID = spaceID;
        cP = checkPoint;
    }

    public void run()
    {
        // compute distance...

        distanceArray.set(sID, manDist);

        try { cP.await(); }
        catch (InterruptedException | BrokenBarrierException ex) {  return; }
    }
}
```

INITIAL TESTING ON WSFM

- 104 locations represented by reference fingerprints
- 800 – 3500 query fingerprints

- Sequential Implementation is approximately 4 – 5 times faster than the initial concurrent implementation.

Hmm… This isn't how concurrency is supposed to work.

LASSONDE
SCHOOL OF ENGINEERING

YORK
UNIVERSITÉ
UNIVERSITY

INITIAL TESTING ON WSDFM

Concurrent implementation is 2 times faster than the sequential implementation.

OK, this is more reasonable, but still underwhelming.

Maybe sorting algorithm is too slow?

Why don't we try a different algorithm…

ALTERNATE CONCURRENT IMPLEMENTATION

1. Allocate arrays to hold all computed distances and their respective indices.
2. Initialize a CyclicBarrier to pause the main thread until distance calculations are complete.
3. Create threads that will each calculate 5 – 20 distances, passing to them the indices of their respective reference fingerprints.
4. Each thread calculates its respective distances then sorts them (keeping track of their reference indices) prior to writing them to its allocated section of the distance array, then it calls await() on the CyclicBarrier.
5. Main thread calls await() after the creation of the distance calculation threads, the CyclicBarrier allows it to proceed once all distance calculation threads have also called await(), distance calculation threads terminate at this point.
6. Main thread iteratively pulls the smallest distance (and its respective index) from all the first distances in each section of the distance array until the *k* smallest distances of the entire array have been pulled.

TESTING THE IMPACT OF SORTING

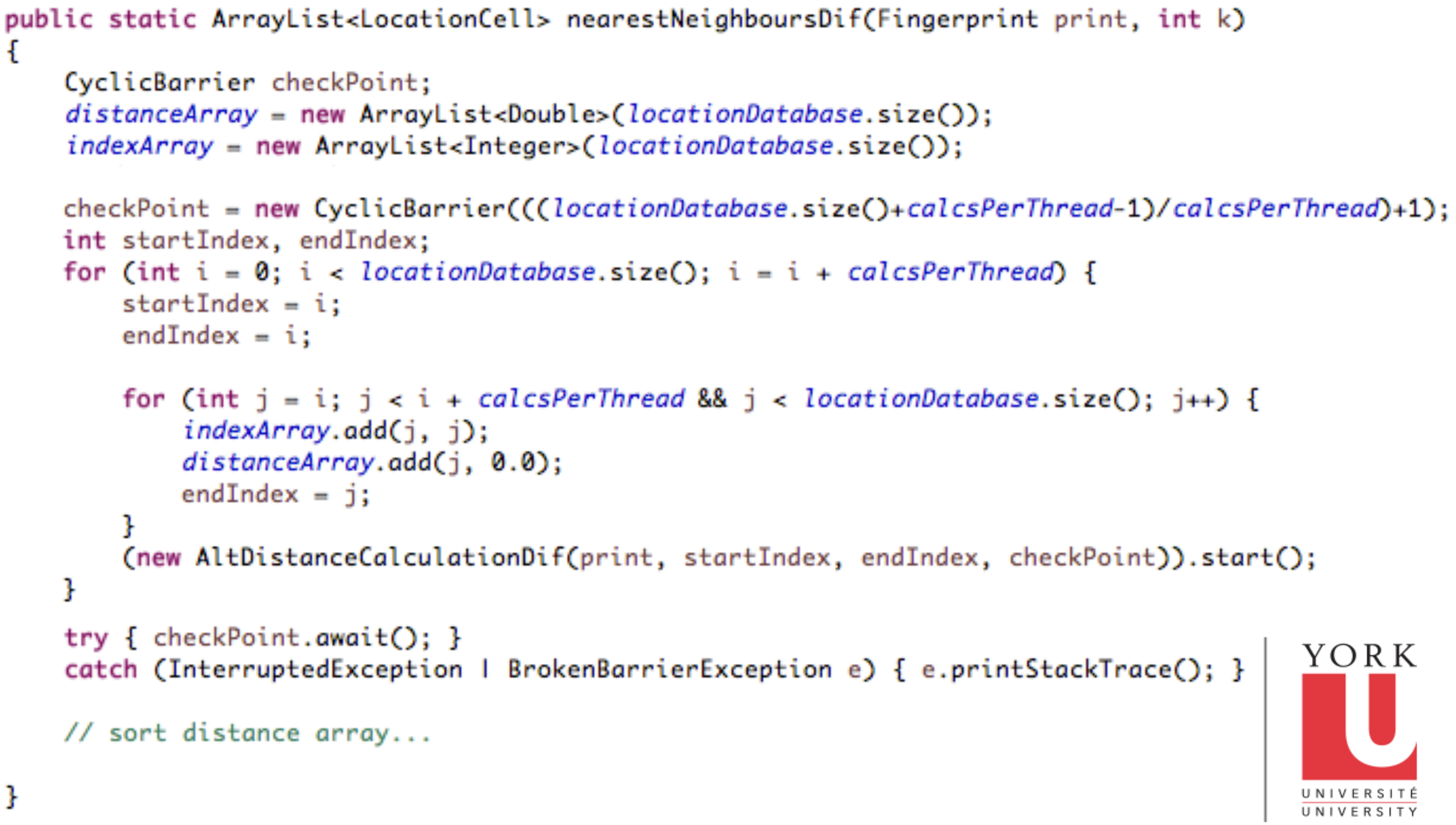
How much is there to gain from improving the sorting?

Not much…

Tests were performed that omitted the sorting component of the k-NN algorithm (returned incorrect results).

Performance gains were negligible!

ALTERNATE CONCURRENT IMPLEMENTATION CODE



```java
public static ArrayList<LocationCell> nearestNeighboursDif(Fingerprint print, int k)
{
    CyclicBarrier checkPoint;
    distanceArray = new ArrayList<Double>(locationDatabase.size());
    indexArray = new ArrayList<Integer>(locationDatabase.size());

    checkPoint = new CyclicBarrier(((locationDatabase.size()+calcsPerThread-1)/calcsPerThread)+1);
    int startIndex, endIndex;
    for (int i = 0; i < locationDatabase.size(); i = i + calcsPerThread) {
        startIndex = i;
        endIndex = i;

        for (int j = i; j < i + calcsPerThread && j < locationDatabase.size(); j++) {
            indexArray.add(j, j);
            distanceArray.add(j, 0.0);
            endIndex = j;
        }
        (new AltDistanceCalculationDif(print, startIndex, endIndex, checkPoint)).start();
    }

    try { checkPoint.await(); }
    catch (InterruptedException | BrokenBarrierException e) { e.printStackTrace(); }

    // sort distance array...

}
```

YORK
UNIVERSITÉ
UNIVERSITY

ALTERNATE CONCURRENT IMPLEMENTATION CODE

```java
private static class AltDistanceCalculationDif extends Thread //implements Runnable
{
    Fingerprint fP; int start, end; CyclicBarrier cP;

    public AltDistanceCalculationDif(Fingerprint print, int startIndex, int endIndex, CyclicBarrier checkPoint)
    {
        fP = print;
        start = startIndex;
        end = endIndex;
        cP = checkPoint;
    }

    public void run()
    {
        for (int i = start; i <= end; i++)
        {
            // compute distances...

             distanceArray.set(i, manDist);
         }
         try { cP.await(); } catch (InterruptedException | BrokenBarrierException ex) { return; }
    }
}
```

THE REAL PROBLEM

Thread creation is very computationally expensive!

What about a balanced amount of distance calculation threads?

Using one thread to calculate 10 distances:

- Sequential WSFM is less than 1.5 times faster than concurrent WSFM
- Concurrent WSDFM is about 1.8 times faster than sequential WSDFM

Still not good enough!

FUTURE WORK

Look at re-using threads to remove the massive overhead of thread creation.

New implementation using Thread Pools.

Thorough and more accurate performance testing on Intel’s Manycore Testing Lab.

END OF PRESENTATION

Thank you for your attention.

Feel free to ask any questions you may have.

LASSONDE
SCHOOL OF ENGINEERING

YORK
UNIVERSITÉ
UNIVERSITY