Introduction to Vectorized Processing in MATLAB with Application to MCL

Omid Aghazadeh

Particle Filters

- Components(Similarity to Genetic Algorithm)

Particle Filter	Genetic Algorithm
Particle	Chromosome(Gene)
Particle Set	Population
Diffusion	Mutation Cross-over
Likelihood function(Update process)	Fitness Function
Re Sampling	Selection

Efficient Implementation of MCL in MATLAB

- Basic Code Optimizations e.g. Pre Allocation
- Avoiding For Loops: Many Particles $\rightarrow$ For Loops over particles will be extremely slow!
  - Vectorized Processing (this session)
  - Mex implementation
  - CUDA Mex implementation (not unless the problem is High Dimensional and the number of particles is HUGE)

Introduction to Vectorized Processing in MATLAB

- Motivation:
  - You don't want to wait 10 seconds for each iteration of MCL to finish! 60 iterations would mean that for every change in parameters you have to wait 10 minutes to see the effects!
  - Example MATLAB codes

(download from here)

Ground Rules!

- Matrices in MATLAB are stored in memory in column major order:
  - 2D matrices

    $\text{size}(A) = [d\ n] \rightarrow A(i,j) = A(i + (j-1)*d)$
  - n-D matrices

    $\text{size}(A) = [d_1\ d_2\ \dots\ d_n] \rightarrow$

    $A(i_1,i_2,\dots,i_n) = A(i_1 + (i_2-1)*d_1 + (i_3-1)*d_1*d_2 +$

    $(i_4-1)*d_1*d_2*d_3 + \dots + (i_n-1)*d_1*\dots*d_{n-1})$

Ground Rules 2

- Vectors better (should!) be stored column wise in matrices:

  $V_1, V_2, \ldots, V_n \rightarrow V = [V_1\ V_2\ \ldots\ V_n]$

  $V(i,j) \rightarrow$ ith dimension of the jth vector

  $\mathrm{size}(V_1) = \ldots = \mathrm{size}(V_n) = [d\ 1] \rightarrow \mathrm{size}(V) = [d\ n]$

- DIMENSIONS! The simplest and most efficient way to find out if something is wrong!

Ground Rules 3

- Multiple entities(e.g. Matrices) better be stacked on the last singleton dimension:

  $E_1, E_2, \ldots, E_n \rightarrow$

  $E = \mathrm{cat}(\mathrm{ndims}(E_1)+1, E_1, E_2, \ldots, E_n)$

  $\mathrm{size}(E_1) = \ldots = \mathrm{size}(E_n) = [d_1\ d_2\ \ldots\ d_D] \rightarrow$

  $\mathrm{size}(E) = [d_1\ d_2\ \ldots\ d_D\ n]$

- The (:) operator reshapes a matrix to a 1D vector(the same order as storage in memory)

Basic Operations on Vectors

- Sorting arrays:
  - [value,index] = sort(V,'ascend')
    - V_asc= V(index); re orders(warps) V in the order of index (ascending) values (value=V_asc)
- Finding indicies
  - index = find(P) e.g. P=[V == max(v)]: finds the nonzero elements of the P vector.
  - The same for matrices:
    - index = find(A(:) == max(A(:)))

Sample Codes

- Mahalanobis Distance (many to one)

$$D_M(x, y, sigma) = (x-y)^T sigma^{-1}(x-y)$$

- Finding the closest points, 2 sets

$$C_2(X, Y) = arg\ min_{x \in X, y \in Y} \|x-y\|^2$$

- Finding the closest points, 3 sets

$$C_3(X, Y, Z) = arg\ min_{x \in X, y \in Y, z \in Z} \left\| (x-y) - \frac{z-y}{\|z-y\|^2}(z-y).(x-y) \right\|^2$$

(download from here)

Now, let's have some fun!

- Consider Matrix Multiplication:

$$C = A * B: \ C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

- Write code to do these operations in one short line(A,B are square and equally sized):

$$W = A ! B: \ W_{i,j} = \sum_k A_{k,i} B_{j,k}$$

$$X = A @ B: \ X_{i,j} = \sum_k A_{i,k} B_{j,k}$$

$$Y = A ' B: \ Y_{i,j} = \sum_k A_{i,j} B_{j,k}$$

$$Z = A \$ B: \ Z_{i,j} = \sum_k A_{N-i,k} B_{N-k,j}$$

Notes

- Vectorized processing can lead to significant speed ups
- It will not always pay to use full vectorized processing, but it almost always pays to vectorize over largest dimension(s) if memory alows it
- Vectorized processing, once mastered, will lead to less code $\longrightarrow$ faster development, less logical errors!