# Justification

After improving the hot three loops of the first few iterations of ML2D, we discovered that the remaining iterations were dominated by the BackProjection effort.

With ML3D the timing is even more distributed

1. 20% getAllSquaredDifferencesFineML
2. 20% backProjection
3. 20% updateModel
4. 40% other

This document describes the loop structure of the first three of these, then shows how to optimize them for a machine where there is insufficient memory bandwidth beyond L2 to keep many cores busy.

# Background material

<http://www2.mrc-lmb.cam.ac.uk/relion/index.php/PreProcessing>

<http://www.sciencedirect.com/science/article/pii/S0022283611012290>

# The Three Nested Loops

## getAllSquaredDifferencesFineML

for every n, class, dir, psi, over\_rot, image, translation, over\_translation

if mlModel.pdf\_class.rptrAll()[iclass] <= 0.0 continue

if mlModel.pdf\_direction[iclass].rptrAll()[idir] <= 0.0 continue

if !significant(class, dir, psi, image, trans) continue

compute the projected rotated frequency (psi, over\_rot, dir, class, n) aka PRF

this requires a complicated formula involving (psi, over\_rot, dir)

compute the translated frequency (trans, over\_trans, image, n) aka TF

this requires a complicated formula involving (trans, over\_trans)

mweight[class, dir, psi, over\_rot, image, translation, over\_translation] +=

f(PRF, TF, Minvsigma2[n])

## backProjection

for every n, class, dir, psi, over\_rot, image, translation, over\_translation

if !significant(class, dir, psi, image, trans) continue

compute the translated frequency

this requires a complicated formula involving (translation, over\_translation)

??? back project into the class dir psi over\_rot

this requires a complicated formula involving (psi, over\_rot, dir)

## updateModel

for every n, class, dir, psi, over\_rot, image, translation, over\_translation

if !significant(class, dir, psi, image, trans) continue

if !significantWeight(mweight[class, dir, psi, over\_rot, image, trans, over\_tran]) continue

inputs:

Fctfs(image, n)

particleModel.Fimages\_mask\_fine(image, n)

PRF(psi, over\_rot, dir, class, n)

this requires a complicated formula involving (dir,psi, over\_rot)

compute the translated frequency (trans, over\_trans, image, n) aka TF

this requires a complicated formula involving (trans, over\_trans)

outputs

thread\_wsum\_norm\_correction\_tid[iimage]

which is later reduced into exp\_wsum\_norm\_correction

# Observation

The assistant tables that speed up rotation and translation are only used a few times (number of over\_rots and number of over\_trans) before being discarded, and once there is only one significant (psi,trans) for a (class,image), they are only used about 4-8 times each.ut 4-8 times each.rans for a class,image pair, this is only about 4-8 times each!  
rots and number of over\_trans) before being

However, there is no reason that the images need be done in ascending order! Instead they should be done in the order that reuses the (class,dir) memory and the (dir,psi,over\_rot) and the (trans,over\_trans) tables.

This could be done in a single pass over the images, putting them into lists indexed by their first significant (class, dir, psi, trans). By doing the images in this order, there would be a lot of reuse!

This is easy to do, by sorting the image numbers by their first significant (class, dir, psi, trans) coordinate.

At a minimum, this should increase the reuse of the (class, dir) plane of data.

This plane of data is currently fetched and rotated before being matched against the translated images, but given than translation and rotation can be combined into a single operation in fourier space, this is maybe not the right thing to do. Maybe the unrotated plane should be kept, and the rotation and translation done during the inner loop.

This is especially plausible if we then to the addition over\_rotations and over\_translations by modifying the aTable and bTable rather than recomputing them from scratch.

This all needs to be prototyped…