Attached

Qfunc.h : headers and prototypes
Qfunc.c : contains general and dummy functions
Qnorm-v0.c : first prototype with a de-coupled structure, processing file by file

To compile: gcc Qnorm-v0.c –o qnorm
or to produce a debugging executable: gcc Qnorm-v0.c –DDEBUG –o qnorm

To execute: Try simply: qnorm –T
It will:

use qInput.txt as list of files to be processed (attached an example)
produce qOut.bin as a by-column binary matrix
-T will transform qOut.bin into qOut.bin.txt a by-column text tabulated file

Athougth I'm including 3 test files f1-15.txt f2 and f3, managing the parameter –E we can execute for 1, 2 or 3 files (or include as much files as you wish)

I have tested the –M default option: managing the Index matrix in memory (in the weekend will try with –D option)

Code Files

Qfunc.h : headers and prototypes
Qfunc.c : contains general and dummy functions
Qnorm.c: prototype

Functions: (Qfunc.c)

struct params *CommandLine(int argc, char *argv[]) analise the command line.

Option	Description	Default value	alternative Values
-p	Number of Nodes	4	not used
-i	File name (list of files)	qInput.txt	valid existing pathname
-o	Output binary matrix	qOut.bin	binary by columns file
-e	Number of experiments	2	positive integer
-g	Number of genes	15	positive integer
-t	Transpose the fileOut	Not	-T (yes) [use only to debug]
-m	Index Matrix in mem/disk	M	D

struct Files* LoadListOfFiles(struct params *) : Load into memory a list of files to be Qnormalised. This file contains one line for each datafile which includes: name, number of genes and type format. In the future we can manage files with different number of genes and different formats

void LoadFile(struct Files*, int, double *); Obtains an array of real values representing the data to normalise Currently I wrote a dummy function to load a text file with one value per line

void QsortC(double *array,int l,int r,int *index); and int partition(double* a, int l, int r, int *indexes); to order values and indexes

Also the following error-handling functions are included

void terror(char *); : print a message in stderr and ends execution
void Alerta(char *,char *): print a warning in stderr
void DebugPrint(char *, double*, int); : print a real values vector with a label
int TransposeBin2Txt(struct params*);: the output is written in disk column by column in binary (to facilitate recording). This function should be used only for testing and transpose the disk matrix and traduce the binary file into a text delimited file (one gene by row with several values

Next functions are in the main program ‘Qnorm.c’ since these are more related to it

void QNormMain(struct params*, struct Files*);: Is the upper level function.I have splited this part to facilitate the parallel code. This function –most probably- will drive the parallel execution (load distribution and synchronization)

Memory demand:

double *dataIn, *dataOut; : two column arrays (nGenes each)

int **mIndex, *dIndex: The full index (integers) matrix is managed in Disk (only a column the dIndex array for indexes is used). If –M parameter is used, the matrix will be managed in memory (mIndex)

struct Average *AvG; // global Average by row

void **AccumulateRow**(struct Average *, double *, int); Decupled function that receive an array of ordered values and accumulate by rows.

void **QNormMain**(struct params *p, struct Files* fList){ // MAIN Qnorm function

```
{
   Memory allocation and initializations

   for (i=0; i< nE; i++) { // Qnorm for each datafile: STEP 1
        LoadFile(fList, i, dataIn);
        Qnorm1(dataIn, dIndex, fList[i].nG);
              // Init correlative index
              // order values and indexes
              // ordered values are returned in the same dataIn
              // Index contains the original position

        AccumulateRow(AvG, dataIn , nG);

        Manage the Index in memory or send it to disk
   }

   for (i=0;i<nG;i++)    // Row average
       AvG[i].Av /=AvG[i].num;

   // produce the ORDERED output file [STEP 2]

   Prepare Output file and one column 'dataOut' array

   for (i=0;i<nE;i++) {
        Get the column index (from memory or from disk)
        for (j=0;j<nG;j++) { // complete the output vector
          dataOut[dIndex[j]]=AvG[j].Av;
          File positioning and writing the vector
        }
   }

   if (p->Traspose) TransposeBin2Txt(p);

}
```