**ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ**
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ

# ΗΡΥ 418 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΠΑΡΑΛΛΗΛΩΝ ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

## ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2018-19

## Άσκηση 2 : Παραλληλισμός με χρήση SIMD εντολών, MPI και PTHREADS

**Εαρινό εξάμηνο 2018-19**
**Ν. Αλαχιώτης**

## Περιγραφή

Στην άσκηση αυτή θα χρησιμοποιήσετε συνδυαστικά Streaming SIMD Extensions (SSE), MPI και Pthreads για να παραλληλοποιήσετε τον υπολογισμό μιας απλοποιημένης μορφής του ω statistic, όπως εφαρμόζεται για ανίχνευση θετικής επιλογής σε ακολουθίες DNA.

Το ω statistic υπολογίζεται ως εξής:

$$num = \frac{L+R}{\left(\frac{m*(m-1.0)}{2.0}\right)+\left(\frac{n*(n-1.0)}{2.0}\right)}$$

$$den = \frac{C-L-R}{m*n}$$

$$\omega = \frac{num}{den+0.01}$$

Οι παραπάνω υπολογισμοί επαναλαμβάνονται επαναληπτικά για ένα σύνολο N DNA θέσεων, για τις οποίες μας ενδιαφέρει ο εντοπισμός της μέγιστης ω τιμής, της ελάχιστης ω τιμής, καθώς και της μέσης ω τιμής. Το N είναι μεταβλητή για την οποία ισχύει $N \geq 1$ . Όλα τα δεδομένα εισόδου είναι τοποθετημένα σε arrays μήκους N (μεταβλητή του χρήστη).

**Εκτέλεση**

1. Αρχικά μελετήστε τον C κώδικα που βρίσκεται στο τέλος της εκφώνησης, τον οποίο μπορείτε να θεωρήσετε ως κώδικα αναφοράς για έλεγχο ορθότητας της τελικής υλοποίησης σας. Ο συγκεκριμένος κώδικας δημιουργεί και αρχικοποιεί 6 arrays, τα mVec, nVec, Lvec, Rvec, Cvec και FVec, με τυχαίες τιμές. Η main αποτελείται από 3 βασικά κομμάτια κώδικα: α) δέσμευση μνήμης, β) αρχικοποίηση και γ) υπολογισμός των ω τιμών. Προσέξτε το loop: `for(unsigned int j=0;j<iters;j++)`, το οποίο επαναλαμβάνει τους υπολογισμούς ώστε να κάνετε πιο ακριβή μέτρηση βάση του μέσου όρου περισσότερων εκτελέσεων.

2. Στη συνέχεια επικεντρωθείτε στο τρίτο κομμάτι της main, δλδ. στον υπολογισμό των ω τιμών, και επιταχύνετε την εκτέλεση του συγκεκριμένου for-loop: `for(unsigned int i=0;i<N;i++)` με την χρήση SSE εντολών (πλάτος λέξης 128 bits). Για να χρησιμοποιήσετε SSE εντολές (εκτός των απαιτούμενων αλλαγών στον κώδικα, library κλπ) πρέπει να χρησιμοποιήσετε το κατάλληλο gcc flag, π.χ., -msse4.2 . Μπορείτε να κάνετε (μικρο)αλλαγές πέρα από τις δηλώσεις μεταβλητών και στον υπόλοιπο κώδικα (εκτός του for-loop), αν το κρίνετε απαραίτητο, οι οποίες πρέπει να αναφερθούν και να σχολιαστούν στην αναφορά σας. Καλείστε να δείξετε και να αξιολογήσετε ξεχωριστά όλα τα βήματα μετατροπής κώδικα που θα ακολουθήσετε μέχρι την εισαγωγή SSE εντολών στον κώδικα, δλδ. τουλάχιστον τα βήματα loop unrolling και jamming.

3. Ως επόμενο βήμα καλείστε να παραλληλοποιήσετε την εκτέλεση του SSE κώδικα με την χρήση pthreads. Τα pthreads θα δημιουργούνται μια φορά στην αρχή της main και θα γίνονται join πριν την επιστροφή της main. Όσο το master thread εκτελεί τα κομμάτια κώδικα για δέσμευση μνήμης και αρχικοποίησης, τα worker threads είναι σε λειτουργία busy-wait. Θεωρήστε ότι το loop: `for(unsigned int j=0;j<iters;j++)` πλέον προσομοιώνει πολλαπλές επαναλήψεις του `for(unsigned int i=0;i<N;i++)` σε μια υποθετική εκτέλεση του κώδικα για πραγματικά δεδομένα. Βάση αυτού, στην αρχή κάθε iteration του loop `for(unsigned int j=0;j<iters;j++)`, το master thread μοιράζει τους υπολογισμούς στα worker threads συμπεριλαμβανομένου και του ίδιου, και συγχρονίζει όλα τα threads στο τέλος του κάθε iteration. Η υλοποίηση σας θα πρέπει να λειτουργεί σωστά για οποιονδήποτε αριθμό από threads. Ο συνολικός αριθμός των threads T που θα τρέχουν κατά τη διάρκεια του παράλληλου κομματιού της υλοποίησης σας είναι παράμετρος του χρήστη. Αξιολογείστε και παρουσιάστε την απόδοση της παραλληλοποίησης σας για 2 και 4 threads.

4. Τέλος καλείστε να δημιουργήσετε έναν αριθμό P από διεργασίες (χρησιμοποιήστε παραλληλισμό με MPI (Message Passing Interface)), όπου το P είναι παράμετρος του χρήστη. Θεωρήστε ότι κάθε διεργασία θα εκτελεστεί σε διαφορετικό node ενός computer cluster με distributed memory. Για ευκολία κατά τον έλεγχο ορθότητας, θεωρήστε ότι τα 6 arrays (mVec, nVec, Lvec, Rvec, Cvec και Fvec) που είναι τα δεδομένα εισόδου και εξόδου για το loop `for(unsigned int i=0;i<N;i++)` που υπολογίζει ω τιμές δημιουργούνται και αρχικοποιούνται από κάθε διεργασία ανεξάρτητα. Για ευκολία στην υλοποίηση μπορείτε να υποθέσετε ότι κάθε node μπορεί να έχει αποθηκευμένα στη μνήμη του όλα τα δεδομένα, αλλά θα επεξεργάζεται μόνο τα δεδομένα που του αντιστοιχούν βάση του διαμοιρασμού των υπολογισμών στις διεργασίες. Κάθε διεργασία θα χρησιμοποιεί T threads για υπολογισμούς (busy-wait λειτουργία), ενώ κάθε thread θα εκτελεί SSE εντολές. Για την συγκεκριμένη υλοποίηση θεωρήστε ότι ο αριθμός των iterations για το loop `for(unsigned int j=0;j<iters;j++)` είναι 1 (μπορείτε να βγάλετε το συγκεκριμένο loop από τον κώδικα σε αυτό το σημείο). Κάθε διεργασία θα δημιουργεί T threads σε busy-wait λειτουργία, θα δεσμεύει την απαραίτητη μνήμη, θα αρχικοποιεί όλα τα arrays με τον ίδιο τρόπο και στη συνέχεια θα αναλαμβάνει μέρος των υπολογισμών. Η διεργασία 0 στο τέλος θα συγκεντρώνει τα αποτελέσματα των υπόλοιπων διεργασιών και θα εκτυπώνει το τελικό αποτέλεσμα. Αγνοείστε την απόδοση του παραλληλισμού σε αυτό το σημείο. Το ζητούμενο είναι η ορθότητα της συνολικής υλοποίησης για τυχαίο αριθμό από διεργασίες και threads.

## ΠΡΟΣΟΧΗ

Τα ζητούμενα 2, 3 και 4 έχουν παρουσιαστεί στην συγκεκριμένη άσκηση ξεκινώντας από το επίπεδο πυρήνα, προς το επίπεδο επεξεργαστή, και στη συνέχεια node. Δεν είναι απαραίτητο να ακολουθηθεί η ίδια σειρά για την υλοποίηση τους. Αν για λόγους ευκολίας κρίνετε ότι σας εξυπηρετεί να προχωρήσετε την υλοποίηση ανάποδα, μπορείτε πρώτα να υλοποιήσετε τις διαφορετικές διεργασίες (MPI) βάση του κώδικα αναφοράς που δίνεται, στη συνέχεια την χρήση pthreads πάλι βάση του κώδικα αναφοράς που σας δίνεται, και τέλος την επιτάχυνση του loop για υπολογισμούς ω τιμών με την χρήση SSE εντολών.

**Πληροφορίες για SIMD εντολές:**
https://software.intel.com/sites/landingpage/IntrinsicsGuide

**Πληροφορίες για MPI:**
http://mpitutorial.com/tutorials/
http://coewww.rutgers.edu/www1/linuxclass2010/lessons/clusters/sec_8.php

Για να χρησιμοποιήσετε MPI (εκτός των απαιτούμενων αλλαγών στον κώδικα, library κλπ) πρέπει να χρησιμοποιήσετε το mpicc (αντί του gcc), να εκτελέσετε την εντολή lamboot, και στη συνέχεια να καλέσετε το executable ως εξής: mpiexec -n P ./executable_name arguments, όπου P είναι ο αριθμός των processes που θα δημιουργηθούν. Αυτό μπορεί να διαφέρει ή να μην χρειάζεται ανάλογα με το σύστημα σας.

**Παραδοτέα:**

1. Source code για τη υλοποίηση με SSE εντολές (τελική + όλες τις ενδιάμεσες υλοποιήσεις)
2. Source code για τη υλοποίηση με SSE εντολές + pthreads
3. Source code για τη υλοποίηση με SSE εντολές + pthreads + MPI
4. Run script που θα καλεί τον reference κώδικα, και όλες τις τελικές υλοποιήσεις (SSE, SEE+PTHREADS, SSE+PTHREADS+MPI) για N = 10000000, και τους 4 συνδυασμούς για P = 2 και 4, και T = 2 και 4.
5. Αναφορά (PDF) που να περιγράφει, να σχολιάζει, και να αξιολογεί τις υλοποιήσεις σας απο άποψη απόδοσης (execution time), εκτός από την υλοποίηση με MPI.

**BONUS 1 [30%]**
**Παρουσίαση, υλοποίηση, και αξιολόγηση διαφορετικού memory layout που να δίνει καλύτερη απόδοση για την υλοποίηση με SSE εντολές. Ακόμη και αν δεν παρατηρήσετε βελτίωση θα πρέπει να δικαιολογήσετε γιατί θα έπρεπε να υπήρχε βελτίωση λόγω του layout.**

**BONUS 2 [20%]**
**Δημιουργία της αναφοράς με τη χρήση Latex (https://www.latex-project.org/) και παράδοση των source files. Μπορείτε να χρησιμοποιήσετε οποιοδήποτε document template σας βολεύει. Η συγγραφή της αναφοράς σε Latex μπορεί να γίνει και online σε private Overleaf project (https://www.overleaf.com/).**

## Ημερομηνία υποβολής: 2 Ιουνίου

**Reference code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <assert.h>
#include <float.h>


#define MINSNPS_B 5
#define MAXSNPS_E 20

double gettime(void);
float randpval (void);

double gettime(void)
{
        struct timeval ttime;
        gettimeofday(&ttime , NULL);
        return ttime.tv_sec + ttime.tv_usec * 0.000001;
}

float randpval (void)
{
        int vr = rand();
        int vm = rand()%vr;
        float r = ((float)vm)/(float)vr;
        assert(r>=0.0f && r<=1.00001f);
        return r;
}

int main(int argc, char ** argv)
{
        assert(argc==2);

        double timeTotalMainStart = gettime();

        float avgF = 0.0f;
        float maxF = 0.0f;
        float minF = FLT_MAX;

        unsigned int N = (unsigned int)atoi(argv[1]);

        unsigned int iters = 10;

        srand(1);

        float * mVec = (float*)malloc(sizeof(float)*N);
        assert(mVec!=NULL);

        float * nVec = (float*)malloc(sizeof(float)*N);
        assert(nVec!=NULL);

        float * LVec = (float*)malloc(sizeof(float)*N);
        assert(LVec!=NULL);

        float * RVec = (float*)malloc(sizeof(float)*N);
        assert(RVec!=NULL);

        float * CVec = (float*)malloc(sizeof(float)*N);
        assert(CVec!=NULL);

        float * FVec = (float*)malloc(sizeof(float)*N);
        assert(FVec!=NULL);

        for(unsigned int i=0;i<N;i++)
        {
                mVec[i] = (float)(MINSNPS_B+rand()%MAXSNPS_E);
                nVec[i] = (float)(MINSNPS_B+rand()%MAXSNPS_E);
                LVec[i] = randpval()*mVec[i];
                RVec[i] = randpval()*nVec[i];
                CVec[i] = randpval()*mVec[i]*nVec[i];
                FVec[i] = 0.0;

                assert(mVec[i]>=MINSNPS_B && mVec[i]<=(MINSNPS_B+MAXSNPS_E));
                assert(nVec[i]>=MINSNPS_B && nVec[i]<=(MINSNPS_B+MAXSNPS_E));
                assert(LVec[i]>0.0f && LVec[i]<=1.0f*mVec[i]);
```

```
                assert(RVec[i]>0.0f && RVec[i]<=1.0f*nVec[i]);
                assert(CVec[i]>0.0f && CVec[i]<=1.0f*mVec[i]*nVec[i]);
        }

        double timeOmegaTotalStart = gettime();
        for(unsigned int j=0;j<iters;j++)
        {
                avgF = 0.0f;
                maxF = 0.0f;
                minF = FLT_MAX;
                for(unsigned int i=0;i<N;i++)
                {
                        float num_0 = LVec[i]+RVec[i];
                        float num_1 = mVec[i]*(mVec[i]-1.0f)/2.0f;
                        float num_2 = nVec[i]*(nVec[i]-1.0f)/2.0f;
                        float num = num_0/(num_1+num_2);

                        float den_0 = CVec[i]-LVec[i]-RVec[i];
                        float den_1 = mVec[i]*nVec[i];
                        float den = den_0/den_1;

                        FVec[i] = num/(den+0.01f);

                        maxF = FVec[i]>maxF?FVec[i]:maxF;
                        minF = FVec[i]<minF?FVec[i]:minF;
                        avgF += FVec[i];
                }
        }
        double timeOmegaTotal = gettime()-timeOmegaTotalStart;
        double timeTotalMainStop = gettime();

        printf("Omega time %fs - Total time %fs - Min %e - Max %e - Avg %e\n",
timeOmegaTotal/iters, timeTotalMainStop-timeTotalMainStart, (double)minF, (double)maxF,
(double)avgF/N);

        free(mVec);
        free(nVec);
        free(LVec);
        free(RVec);
        free(CVec);
        free(FVec);
}
```