ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ & ΥΛΙΚΟΥ

ΗΡΥ 418 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΠΑΡΑΛΛΗΛΩΝ ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2017-18

Άσκηση 2 : Παραλληλισμός με χρήση SIMD εντολών και MPI

Εαρινό εξάμηνο 2017-18
Ν. Αλαχιώτης

Περιγραφή

Στην άσκηση αυτή αρχικά θα χρησιμοποιήσετε Streaming SIMD Extensions (SSE) εντολές για να επιταχύνετε τον υπολογισμό μιας απλοποιημένης μορφής του ω statistic, όπως εφαρμόζεται για ανίχνευση θετικής επιλογής σε ακολουθίες DNA.

Θεωρήστε τους ακόλουθους τύπους:

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

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

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

Οι παραπάνω υπολογισμοί επαναλαμβάνονται επαναληπτικά για ένα σύνολο N DNA θέσεων και μας ενδιαφέρει ο εντοπισμός της μέγιστης ω τιμής. Το N είναι μεταβλητή για την οποία ισχύει $N \geq 1$. Για ευκολία, όλα τα δεδομένα εισόδου είναι τοποθετημένα σε arrays μήκους N. Μελετήστε τον παρακάτω κώδικα, τον οποίο μπορείτε να χρησιμοποιήσετε ως βάση για την υλοποίηση σας αλλά και για έλεγχο ορθότητας. Στη συνέχεια υλοποιήστε και αντικαταστήστε μόνο το **for-loop** που κάνει τους υπολογισμούς ω και βρίσκει τη μέγιστη τιμή με SSE εντολές (128-bit). Μπορείτε να κάνετε (μικρο)αλλαγές και στον υπόλοιπο κώδικα (εκτός του **for-loop**), αν το κρίνετε απαραίτητο, οι οποίες πρέπει να αναφερθούν και να σχολιαστούν στην αναφορά. Για να χρησιμοποιήσετε SSE (εκτός των απαιτούμενων αλλαγών στον κώδικα, library κλπ) πρέπει να χρησιμοποιήσετε το κατάλληλο gcc flag, π.χ., -msse4.2.

Πληροφορίες για SIMD εντολές:

https://software.intel.com/sites/landingpage/IntrinsicsGuide

Reference code:

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

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

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

int main(int argc, char ** argv)
{
	int N = atoi(argv[1]);

	int iters = 1000;

	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(int i=0;i<N;i++)
	{
		mVec[i] = (float)(2+rand()%10);
		nVec[i] = (float)(2+rand()%10);
		LVec[i] = 0.0;
		for(int j=0;j<mVec[i];j++)
		{
			LVec[i] += randpval();
		}
		RVec[i] = 0.0;
		for(int j=0;j<nVec[i];j++)
		{
			RVec[i] += randpval();
		}
		CVec[i] = 0.0;
		for(int j=0;j<mVec[i]*nVec[i];j++)
		{
			CVec[i] += randpval();
		}
		FVec[i] = 0.0;

		assert(mVec[i]>=2.0 && mVec[i]<=12.0);
		assert(nVec[i]>=2.0 && nVec[i]<=12.0);
		assert(LVec[i]>0.0 && LVec[i]<=1.0*mVec[i]);
		assert(RVec[i]>0.0 && RVec[i]<=1.0*nVec[i]);
		assert(CVec[i]>0.0 && CVec[i]<=1.0*mVec[i]*nVec[i]);
	}
```

```
        float maxF = 0.0f;
        double timeTotal = 0.0f;
        for(int j=0;j<iters;j++)
        {
                double time0=gettime();
                for(int i=0;i<N;i++)
                {
                        float num_0 = LVec[i]+RVec[i];
                        float num_1 = mVec[i]*(mVec[i]-1.0)/2.0;
                        float num_2 = nVec[i]*(nVec[i]-1.0)/2.0;
                        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+0.01);

                        maxF = FVec[i]>maxF?FVec[i]:maxF;
                }
                double time1=gettime();
                timeTotal += time1-time0;
        }

        printf("Time %f Max %f\n", timeTotal/iters, maxF);

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

Στη συνέχεια χρησιμοποιήστε παραλληλισμό με MPI (Message Passing Interface) για να δημιουργήσετε P processes και να επιταχύνετε την εκτέλεση του υλοποιημένου με SSE εντολές **for-loop.** Για την εκτέλεση με MPI θεωρήστε ότι όλα τα processes δημιουργούν όλα τα δεδομένα εισόδου (arrays μεγέθους N: `mVec`, `nVec`, `LVec`, `RVec`, `Cvec`) και υπολογίζουν μόνο μέρος των αποτελεσμάτων (μέρος του array `Fvec` ανα process). Περιορίστε την ανταλλαγή δεδομένων μεταξύ των processes στην ελάχιστη δυνατή ώστε το process 0 να είναι σε θέση να εκτυπώσει στην οθόνη την σωστή μέγιστη ω τιμή για το σύνολο των N DNA θέσεων.

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.

Πληροφορίες για MPI:

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

Παραδοτέα:

1. Source code για τη υλοποίηση με SSE εντολές (μόνο την τελική υλοποίηση)
2. Source code για τη υλοποίηση με SSE εντολές + MPI
3. Run script που θα καλεί τον reference κώδικα, την υλοποίηση με SSE, και την υλοποίηση με SSE+MPI, για N=100, 1000, 10000, 100000, και P=2, 4, 8.
5. Αναφορά (PDF) που να περιγράφει, να σχολιάζει, και να αξιολογεί τις υλοποιήσεις σας απο άποψη απόδοσης (execution time). Για την υλοποίηση με SSE να δείξετε και να σχολίασετε τις ενδιάμεσες τροποποιήσεις του for-loop.