Index

Objectius

Aquesta pràctica té com a objectius introduir els conceptes bàsics del model de programació CUDA. Per a la seva realització es posaran en pràctica varis dels conceptes presentats en aquesta assignatura.

Presentació de la pràctica

Cal presentar un document amb les respostes a les preguntes formulades. Per a la PAC es faran servir extensions associades CUDA.

Restriccions de l'entorn

En aquesta PAC utilitzarem l'entorn d'execució Ocelot, un emulador de GPU, que fa possible provar codi sense disposar d'una GPU física. En qualsevol cas, si es disposa d'una GPU podeu validar el vostre codi per a aquesta i podeu realitzar execucions reals pel vostre compte (opcional – no s'espera proporcionar suport).

Material per a la realització de la pràctica

No cal material específic per aquesta PAC més enllà dels materials de l'assignatura. Si decidiu realitzar la implementació per a una GPU real s'espera que feu una cerca per vosaltres mateixos sobre la instal·lació i configuració dels SDK requerit. Podeu desenvolupar un debat al fòrum de l'assignatura per a compartir qualsevol experiència relacionada que desitgeu.

1. «Hello World» en CUDA

A continuació es troba un codi d'exemple per CUDA, en concret una versió del «Hello World» en CUDA:

```
Codi (fitxer hello.cu):
#include <stdio.h>

const int N = 16;
const int blocksize = 16;

__global__
void hello(char *a, int *b)
{
        a[threadIdx.x] += b[threadIdx.x];
}
int main()
{
        char a[N] = "Hello \0\0\0\0\0\0";
        int b[N] = {15, 10, 6, 0, -11, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

        char *ad;
        int *bd;
        const int csize = N*sizeof(char);
        const int isize = N*sizeof(int);

        printf("%s", a);

        cudaMalloc((void**)&ad, csize );
        cudaMalloc((void**)&bd, isize );
```

```
        cudaMemcpy(ad, a, csize, cudaMemcpyHostToDevice );
        cudaMemcpy(bd, b, isize, cudaMemcpyHostToDevice );

        dim3 dimBlock( blocksize, 1 );
        dim3 dimGrid( 1, 1 );
        hello<<<dimGrid, dimBlock>>>(ad, bd);
        cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
        cudaFree( ad );

        printf("%s\n", a);
        return EXIT_SUCCESS;
}
```

Pregunta 1

Descriu el funcionament del programa, és a dir, com s'aconsegueix escriure «Hello World» a partir dels vectors d'entrada i el kernel proporcionat.

Primer s'imprimeix per pantalla la paraula «Hello».

Després s'assigna memòria global del dispositiu GPU per a les matrius.

```
cudaMalloc((void**)&ad, csize );
cudaMalloc((void**)&bd, isize );
```

Es copien les matrius d'entrada (a i b) de la memòria del host a la memòria del dispositiu GPU a l'adreça on indiquen els punters (ad i bd). El paràmetre 'cudaMemcpyHostToDevice' serveix per per especificar origen i destí de la copia.

```
cudaMemcpy(ad, a, csize, cudaMemcpyHostToDevice );
cudaMemcpy(bd, b, isize, cudaMemcpyHostToDevice );
```

Es configuren les dimensions del grid i blocs de fluxes. El grid amb els paràmetres on blocksize és 16 que en aquest cas serà el número de fluxes.

```
dim3 dimBlock( blocksize, 1 );
dim3 dimGrid( 1, 1 );
```

S'invoca el kernel (hello) per generar el grid amb els diferents fluxes que executarà el nucli en el dispositiu GPU.

```
hello<<<dimGrid, dimBlock>>>(ad, bd);
```

El codi del kernel, el que fa es sumar la matriu 'a' els diversos valors que hi ha a la matriu 'b', que pel resultat es transformarà en «World!».

```
a[threadIdx.x] += b[threadIdx.x];
```

Es copia el resultat emmagatzemat al punter 'ad' de la memòria del dispositiu GPU a la matriu 'a' de la memòria del host.

```
cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );
```

S'allibera espai de memòria del dispositiu GPU.

```
cudaFree( ad );
```

Finalment s'imprimeix per pantalla el resultat el resultat de la matriu 'a'.

```
printf("%s\n", a);
```

Pregunta 2

Quina transferència de dades es produeix entre el host i device (GPU)?

Es produeixin dos transferències de dades entre el host i el dispositu GPU:

1. Quan es copien les matrius d'entrada (a i b) de la memòria del host a la memòria del dispositiu GPU a l'adreça on indiquen els punters (ad i bd).

cudaMemcpy(ad, a, csize, cudaMemcpyHostToDevice);
cudaMemcpy(bd, b, isize, cudaMemcpyHostToDevice);

2. Quan es copia el resultat emmagatzemat al punter 'ad' de la memòria del dispositiu GPU a la matriu 'a' de la memòria del host.

cudaMemcpy(a, ad, csize, cudaMemcpyDeviceToHost);

2. Execució mitjançant emulador Ocelot (instruccions)

Ocelot un entorn que permet executar programes CUDA en GPUs o en processadors convencionals. Podeu trobar-hi informació a la següent adreça:

http://code.google.com/p/gpuocelot/

A continuació es detallen les instruccions bàsiques que heu de seguir per utilitzar Ocelot; per a la versió del «Hello World» per GPU (des del node de login):

1. Codi (fitxer hello.cu):
2. Modificar el LD_LIBRARY_PATH adequadament
export
LD_LIBRARY_PATH=/export/apps/gcc/4.8.2/lib:/export/apps/gcc/4.8.2/lib64:/e
xport/apps/ocelot/lib/:/export/apps/boost/lib/:$LD_LIBRARY_PATH
3.
Compilació del codi CUDA
/export/apps/cuda/5.5/bin/nvcc -cuda hello.cu -I /export/apps/ocelot/include/ocelot/api/interface/ -arch=sm_20

4.
El pas anterior generarà un fitxer anomenat hello.cu.cpp.ii que s'haurà de compilar amb g++ per a obtindré un binari executable.
/export/apps/gcc/4.8.2/bin/g++ -o hello hello.cu.cpp.ii -I /export/apps/ocelot/include/ocelot/api/interface/ -L /export/apps/ocelot/lib/ -locelot

5.Executar el programa
./hello
Podeu afegir la modificació del LD_LIBRARY_PATH i el PATH en el vostre .bashrc per a simplificar aquestes instruccions.
L'execució per defecte retorna warnings:

==Ocelot== WARNING: Failed to find 'configure.ocelot' in current directory,
loading defaults.
==Ocelot== INFO: You may consider adding one if you need to change the
Ocelot target, or runtime options.
Podeu ignorar aquests warnings o
"configure.ocelot" amb, per exemple:

{
"ReRoutes": [],
"GlobalConfiguration": {}
}

Tasques realitzades:

He canviat el export per share:
[capa20@eimtarqso pac4]$ export LD_LIBRARY_PATH=/share/apps/gcc/4.8.2/lib:/share/apps/gcc/4.8.2/lib64:/share/apps/ocelot/lib/:/share/apps/boost/lib/:$LD_LIBRARY_PATH

He generat el fitxer CUDA:
[capa20@eimtarqso pac4]$ /export/apps/cuda/5.5/bin/nvcc -cuda hello.cu -I /export/apps/ocelot/include/ocelot/api/interface/ -arch=sm_20

He compilat el fitxer executable a partir del fitxer CUDA:
[capa20@eimtarqso pac4]$ /export/apps/gcc/4.8.2/bin/g++ -o hello hello.cu.cpp.ii -I /export/apps/ocelot/include/ocelot/api/interface/ -L /export/apps/ocelot/lib/ -locelot

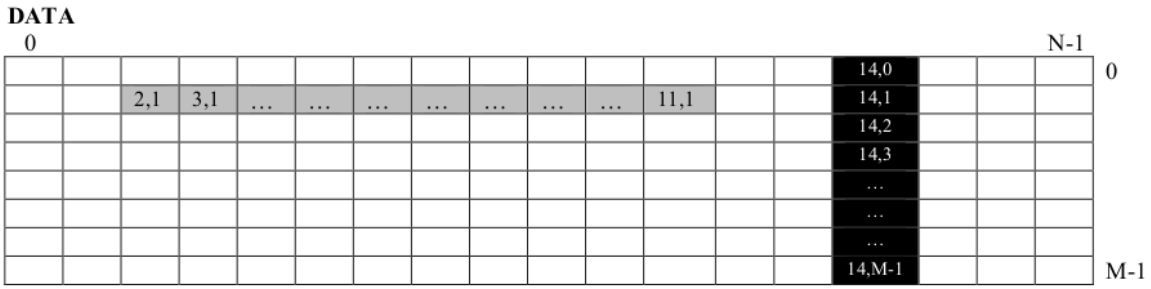
He executat el codi amb el resultat esperat:
[capa20@eimtarqso pac4]$ ./hello
Hello World!

Instruccions per a GPUs físiques (opcional, sense suport)

Podeu trobar informació de com instal·lar i utilitzar els SDK corresponents a CUDA o OpenCL (per GPUs NVIDIA o ati, respectivament) podeu utilitzar els links següents com a referència:
http://developer.nvidia.com/cuda-downloads i http://www.khronos.org/

3. Exercici de programació

Es demana que programeu mitjançant CUDA el següent problema.
A partir d'una matriu DATA[N,M] de N columnes y M files, el programa ha de proporcionar:
1) Una matriu de sortida MOV, on cada punt MOV[i, j] es la mitjana de DATA[i-9, j], DATA[i-8, j], ..., DATA[i, j] (representa una "moving average").
2) Un vector AVG amb la mitjana dels valors de totes les files per a cada columna, és a dir AVG[i] és la mitjana de DATA[i, 0], DATA[i, 1], ..., DATA[i, M-1] tal i com es mostra en la següent il·lustració:



Per exemple:
MOV[11,1] = mitjana dels valors marcats en gris
AVG[14] = mitjana dels valors marcats en negre

Pregunta 3

Proporcioneu la teva implementació.

Per a cada valor de la matriu d'entrada DATA, s'actualitzarà amb la mitjana dels veïns anteriors, on el número de veïns anteriors vindrà donat per la variable ELE. A les primeres columnes on no hi ha suficients veïns anteriors, es mantindrà el valor d'entrada encara que també es podria realitzar la mitjana amb els valors existents modificant la condició al for().

```
[capa20@eimtarqso pac4]$ cat pac4_ex3.cu
#include <stdio.h>
#define N 10 // Nº Columnes
#define M 5  // Nº Files
#define ELE 4 // Elements anteriors

__global__ void moving_average(float *a, float *b) {

        int index = blockDim.x * blockIdx.x + threadIdx.x;
        float result;

        if (index % N >= ELE){ //El primer valor de cada fila a calcular és el Nº ELE
                for(int i=0;i < ELE;i++) {
                result = result + a[index-i];
                }
                b[index] = result/ELE;
        } else {  b[index] = a[index];
        }
}

__global__ void average_col(float *a, float *b) {

        int index = blockDim.x * blockIdx.x + threadIdx.x;
        int column = threadIdx.x;  //  Tenim un thread per columna segons definit a DimBlock (ThreadsPerBlock)
        float result;

        // Recorrer els valors en vertical de la matriu DADES i fer mitjana
        // Per a cada columna (column = nº threads per simplificar) es suma el valor de totes les files (N*i)
        for(int i=0; i<M; i++){
                result = result + a[N*i+column];
        }
        b[index] = result/M;  // Es fa la mitjana amb el nº de files
}


int main(){
        // Definim els punters de les matrius/vector (DADES, MOV i AVG) al host i device
        float *host_data, *host_mov, *host_avg;
        float *device_data, *device_mov, *device_avg;

        const int size_host_data = N*M*sizeof(float);
        const int size_host_mov = N*M*sizeof(float);
        const int size_host_avg = N*sizeof(float);

        // Assignem memòria per a les matrius/vector al host i device
        host_data = (float*)malloc(size_host_data);
        host_mov = (float*)malloc(size_host_mov);
        host_avg = (float*)malloc(size_host_avg);

        cudaMalloc((void**)&device_data, size_host_data);
        cudaMalloc((void**)&device_mov, size_host_mov);
        cudaMalloc((void**)&device_avg, size_host_avg);
```

```
        // Inicialitzem les matrius
        // Matriu DATA[N,M] --> host_data --> Valors aletaris del 0 al 10
        // Matriu MOV[N,M] --> host_mov --> '0'
        // Vector AVG[N] --> host_avg --> '0'
        int i, j;

        for(i=0;i<N*M;i++){ host_data[i]= (float)(rand() % 10);}
        for(i=0;i<N*M;i++){ host_mov[i]= 0;}
        for(j=0;j<N;j++){ host_avg[j]= 0;}

        // Mostrar matriu DADES
        printf("Matriu DADES[N,M]:\n");
        for(int i=0;i<M;i++){
                printf("Row Nº%i |",i);
                for(int j=0;j<N;j++){
                        printf(" %f |",host_data[i*N+j]);
                }
         printf("\n");
        }

        // Copiem les matrius/vector del host al device
        cudaMemcpy( device_data, host_data, size_host_data, cudaMemcpyHostToDevice);
        cudaMemcpy( device_mov, host_mov, size_host_mov, cudaMemcpyHostToDevice);
        cudaMemcpy( device_avg, host_avg, size_host_avg, cudaMemcpyHostToDevice);

        // Definim el nº de blocs, el nº de fluxos i s'invoca el kernel per calcular el 'Moving Average'
        // Creem un únic bloc i un thread per a cada
        int blocksPerGrid = 1; // Nº de blocs
        int threadsPerBlock = N * M; // Nº de blocs
        moving_average<<<blocksPerGrid, threadsPerBlock>>>(device_data, device_mov);

        cudaMemcpy( host_mov, device_mov, size_host_mov, cudaMemcpyDeviceToHost );
        cudaFree( host_mov);

        // Definim el nº de blocs, el nº de fluxos i s'invoca el kernel per calcular el          'Average Column'
        // La matriu dades encara està al dispositiu
        // Creem un únic bloc i un thread per cada columna
        blocksPerGrid = 1;
        threadsPerBlock = N;
        average_col<<<blocksPerGrid, threadsPerBlock>>>(device_data,device_avg);
        cudaMemcpy( host_avg, device_avg, size_host_avg, cudaMemcpyDeviceToHost );
        cudaFree( host_data);
        cudaFree( host_avg);

        // Mostrar matriu MOV
        printf("Matriu MOV[i,j]:\n");
        for(int i=0;i<M;i++){
                printf("Row Nº%i |",i);
                for(int    j=0;j<N;j++){
                        printf(" %f |",host_mov[i*N+j]);
                }
        printf("\n");
        }
        // Mostrar vector AVG
        printf("Matriu AVG[j]:\n");
        printf("Average: ");
                for(int i=0;i<N;i++){
                printf(" %f |",host_avg[i]);
        }
        printf("\n");

        return EXIT_SUCCESS;
}
```

Pregunta 4

Proporcioneu un exemple d'ús del vostre codi. Per exemple podeu generar una matriu d'entrada d'una grandària petita (1000x10) amb valors aleatoris (o seguint una distribució determinada) de tal manera que pugueu generar un gràfic de les dades d'entrada (gràfic amb 10 sèries de 1000 punts), i la matriu i vector de sortida.

He creat un script 'pac4_ex4.sh' per automatitzar les proves. El resultat de l'execució del programa es desa al fitxer *'pac4_ex4_restult.csv'.*

[capa20@eimtarqso pac4]$ cat pac4_ex4.sh

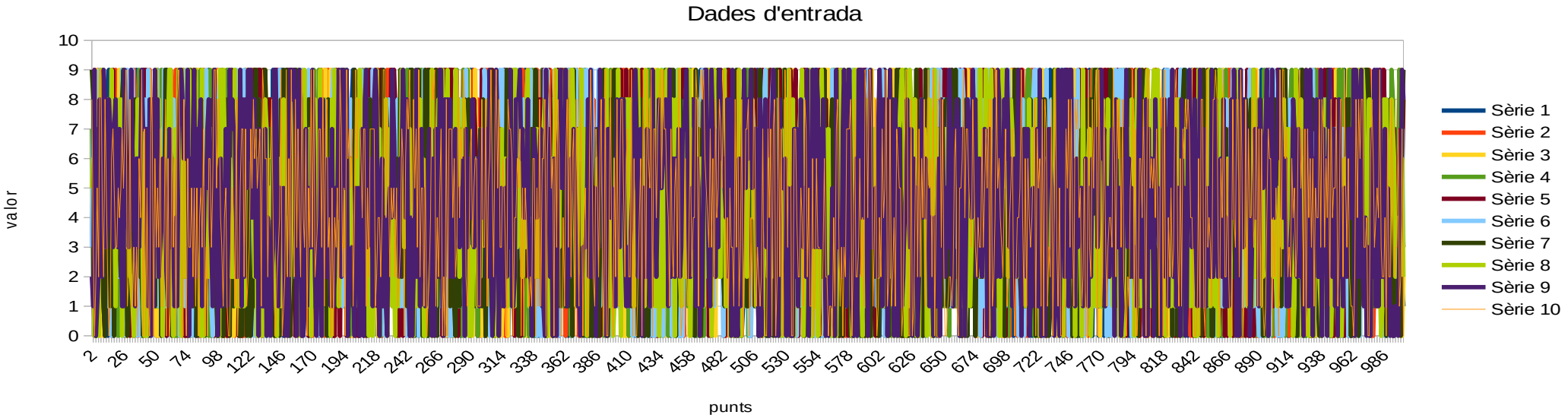
```
export LD_LIBRARY_PATH=/share/apps/gcc/4.8.2/lib:/share/apps/gcc/4.8.2/lib64:/share/apps/ocelot/lib/:/share/apps/boost/lib/:$LD_LIBRARY_PATH

/export/apps/cuda/5.5/bin/nvcc -cuda pac4_ex3.cu -I /export/apps/ocelot/include/ocelot/api/interface/ -arch=sm_20

/export/apps/gcc/4.8.2/bin/g++ -o pac4_ex3 pac4_ex3.cu.cpp.ii -I /export/apps/ocelot/include/ocelot/api/interface/ -L /export/apps/ocelot/lib/ -locelot

./pac4_ex3 >> pac4_ex4_result.csv
```

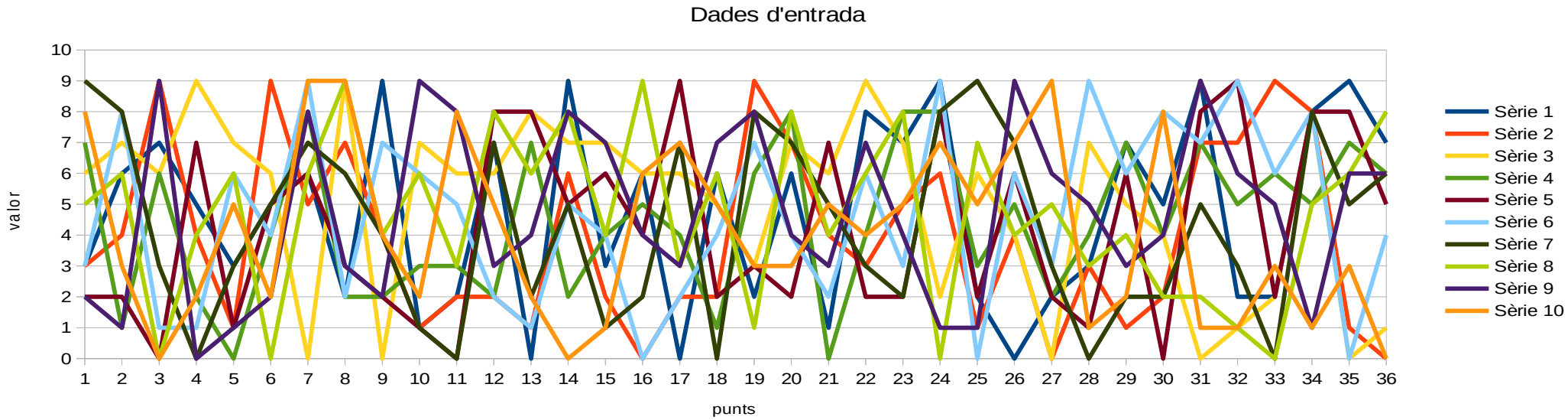
Les comendes serveixen per:

- *Definir les rutes necessàries per generar i compilar el programa.*
- *Generar fitxer CUDA.*
- *Compilar el fitxer executable a partir del fitxer CUDA:*
- *S'executa el programa i desar el resultat en el fitxer 'pac4_ex4_restult.csv'.*

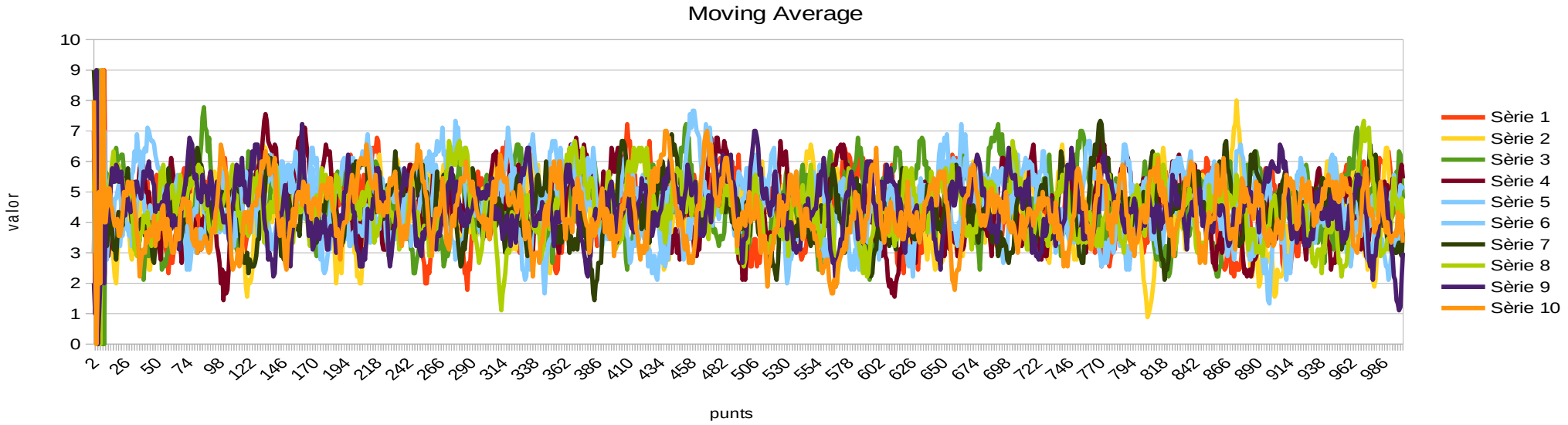
El resultat desat al fitxer 'pac4_ex4_restult.csv' l'he importat a una fulla de càlcul i he generat les següents gràfiques.

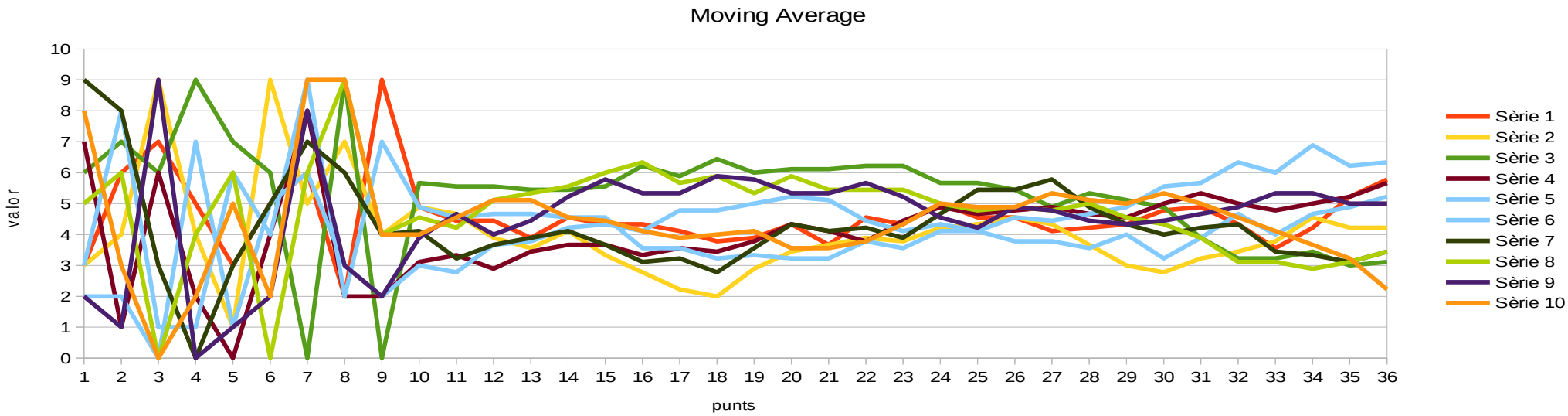
El valors de la matriu d'entrada estan repartits de 0 a 10, degut a que s'ha assignat valors aleatoris i per tant no segueix cap tipus de distribució.



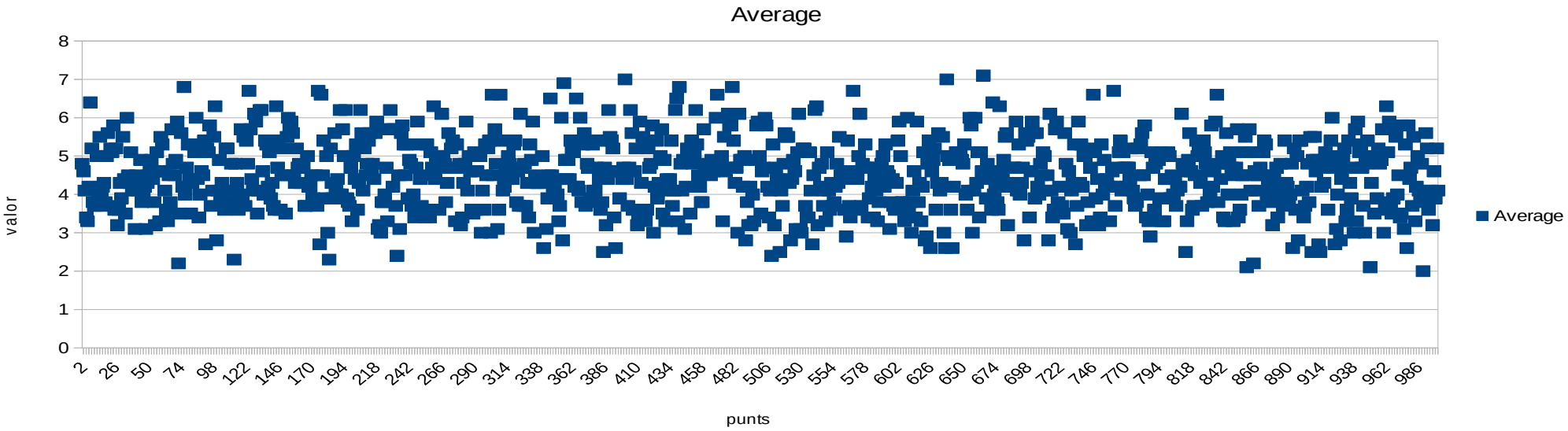


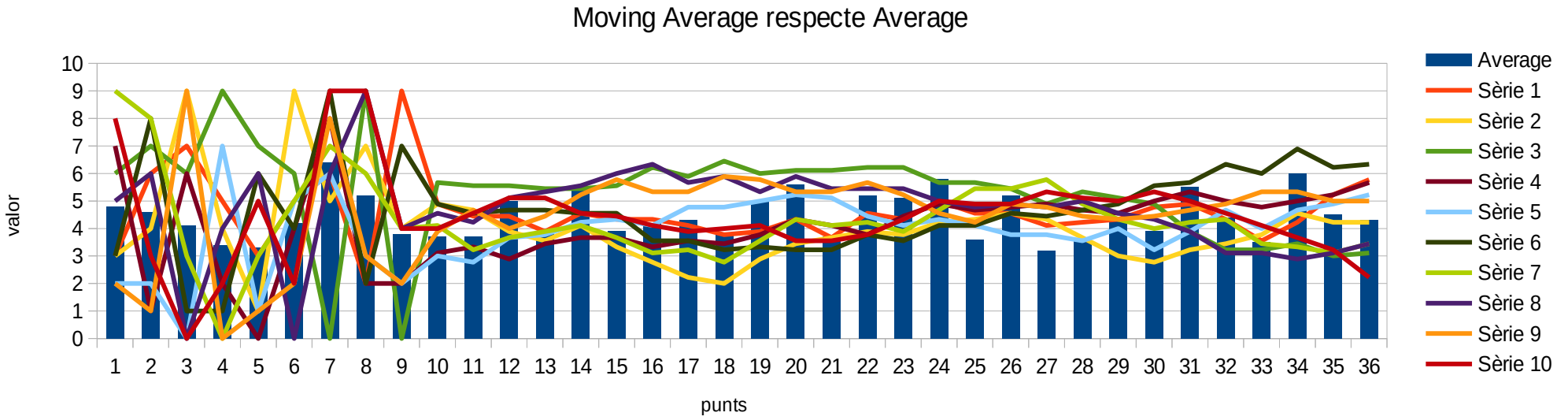
El ‘Moving Average’ s’ha calculat amb la mitjana del valor actual i els 8 veïns anteriors, el resultat són unes sèries més suavitzades amb valors majoritàriament compresos entre 3 i 6, encara que les sèries són diferents entre elles. Els primers 9 valors es manté el valor d’entrada segons s’ha programat.





A la mitjana de les diferents sèries s'observa que els valors resultants estan compresos majoritàriament entre el 3 i 6. Si es compara el 'Movement average' amb la mitjana de les sèries s'observa una similitud en la magnitud dels valors, encara que el 'Average' de les sèries és més variable que el 'Movement average'.





4. Preguntes addicionals (opcional)

En aquesta part de la PAC s'espera que programeu el problema de l'exercici de programació com un problema sintètic. En uns dies se us facilitarà informació per contextualitzar aquest problema aplicant-lo a un problema real relacional amb cyber- infraestructura (part de la PAC1).

Es formularan unes preguntes addicionals (opcionals) que es tindran en compte de cara a l'avaluació.