Processadors MultiThread-Multicore:

Aplicació multithread

La pràctica consisteix a analitzar el comportament d'una aplicació multithread quan s'executa en màquines amb capacitat d'executar diversos threads en paral·lel. Concretament, es vol estudiar la capacitat de càlcul, la productivitat i l'escalabilitat d'un determinat algorisme en màquines multicore.

Es disposarà d'un algorisme seqüencial al qual, mitjançant la llibreria pthreads de POSIX, es paral·lelitzarà creant múltiples threads que s'executen en paral·lel i/o concurrentment.

Les maquines a estudiar són dos servidors amb diferents característiques de potència i consum. El primer (teen) té 2 processadors Intel Xeon E5-2690 a 2.9 GHz amb 8 cors i 2 threads per core cadascun, una memòria RAM de 32GB i un TDP de 135W + 135W. El segon (orca) té un processador AMD Ryzen Threadripper PRO 3995WX a 2.7Ghz amb 64 cors i 2 threads per core, una memòria RAM de 128GB i un TDP de 280W..

Comentaris

- La pràctica es realitzarà en GRUPS DE 2 PERSONES
- Es realitzarà una entrevista amb tots els integrants del grup a la sessió de laboratori que tenen assignada.
- L'informe (obligatòriament en PDF), juntament amb el vídeo de cada membre del grup i el codi implementat, es comprimiran en un únic fitxer ZIP i es guardarà al moodle abans de fer l'entrevista.

Especificació

La tasca d'aquesta pràctica consisteix a paral·lelitzar un codi seqüencial mitjançant threads usant les crides de la llibreria pthread_create i pthread_join (ja explicades en assignatures com a FSO).

El codi proporcionat permet ser paral·lelitzat al primer nivell del bucle. Així no cal fer cap estudi de dependències ja que permet dividir les diferents iteracions del bucle entre els threads que es vulguin crear. Per exemple un bucle de N iteracions i controlat per la variable num es podria dividir entre 4 threads de la següent forma:

```
1. for(num= 0; num < N/4 ;num++)
2. for(num= N/4; num < N/2 ;num++)
3. for(num= N/2; num < 3*N/4 ;num++)
4. for(num= 3*N/4; num < N ;num++)
```

Encara que l'exemple sigui de 4 threads, cal programar el codi perquè admeti un nombre arbitrari de threads, que s'especificarà amb un argument d'entrada.

L'algoritme s'executarà a cadascuna de les dues màquines variant la mida de les dades i el nombre de threads que l'estan executant. S'obtindrà el temps d'execució de cadascuna de les alternatives, per poder fer comparatives després del temps d'execució i de l'speedup respecte a la versió seqüencial més lenta.

El nombre de threads serà de 2, 4, 8, 16, 32, 64, 128 i 256. Cal esperar que la màquina de 2 cpus x 8 cors x 2 threads vegi cert grau de saturació a partir de 16-32 threads. D'altra banda, la màquina de 64 cors x 2 threads és d'esperar que millori fins als 64 threads, i segueixi millorant de manera més continguda fins als 128 threads i finalment vegi cert grau de saturació a partir d'aquest valor.

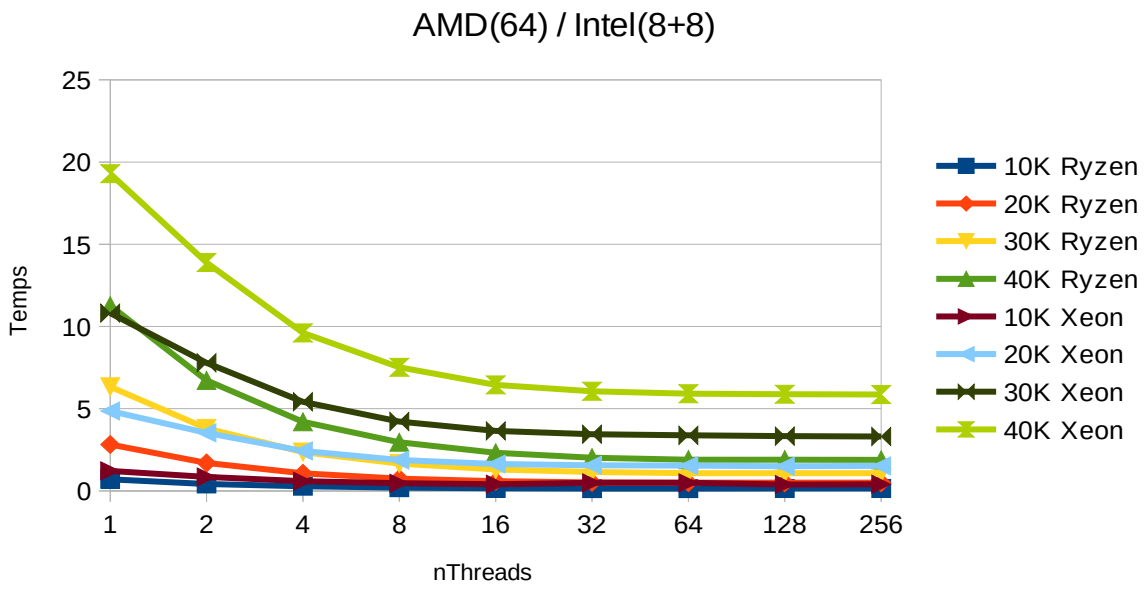
L'algorisme també incrementarà les dades que manipula i calcula. D'aquesta manera es veurà el comportament de l'escalabilitat en funció de la mida del problema. Es consideraran diverses mides i s'inclouran a les diferents gràfiques.

Els resultats es mostraran en gràfics.

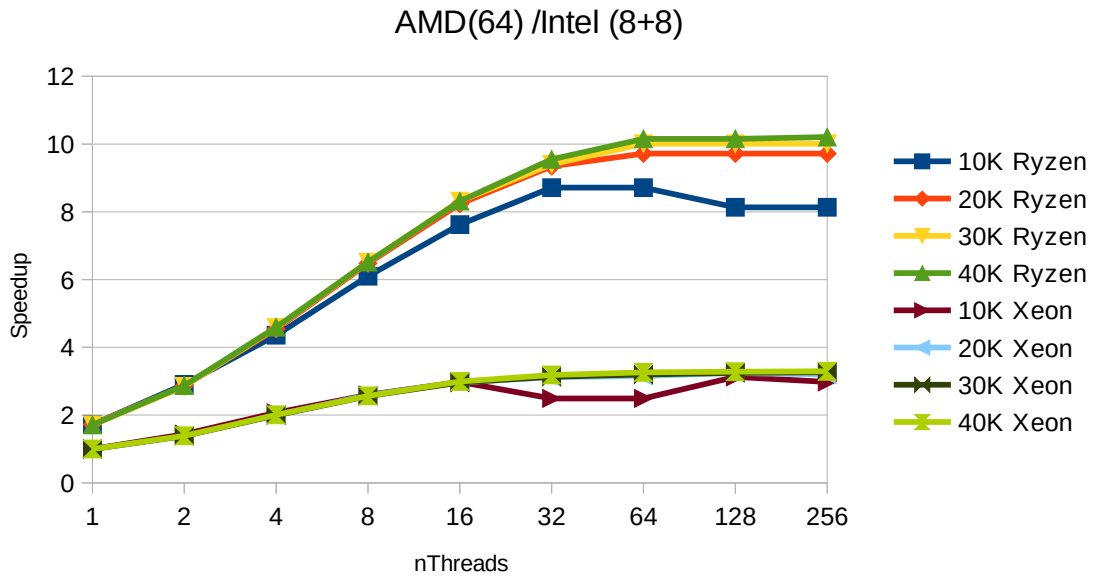
Per als diferents estudis que s'han de realitzar, i si no s'indica el contrari, tingueu en compte els comentaris següents:

1) Es mostrarà una gràfica amb el temps d'execució real per a cadascuna de les mides del problema. On a l'eix de les Y estarà el temps d'execució amb escala lineal o

logarítmica (segons s'apreciïn millor els valors) i a l'eix de les X les diferents execucions que s'han fet variant el nombre de threads.



2) De la mateixa manera, es mostrarà una altra gràfica amb l'speedup relatiu a la versió seqüencial més lenta. En aquest cas la versió seqüencial de l'Intel, on a l'eix de les Y hi haurà l'speedup i a l'eix de les X les diferents execucions que s'han fet variant el nombre de threads.



3) Guia de paral·lelització:

1. Abans de paral·lelitzar i amb el codi seqüencial, estudieu mitjançant la funció clock(), el temps d'execució de cada bucle. Apliqueu la llei d'Amdahl per 2 fins a 256 i infinits processadors.
2. Calculeu la porció que correspondria a cada thread del bucle/s que aneu a paral·lelitzar (N/nTheads).

3. Creeu una funció que serà executada per cada thread de manera que a partir del valor que se li passa, sàpiga que porció de 'num' li correspon.
4. Executeu un bucle de creació de tots els threads.
5. Executeu un bucle d'espera de finalització de tots els threads.
6. El codi a paral·lelitzar és la cerca d'un determinat nombre primer anterior a un valor. Sent N 100M, 100.000.000 → el darrer primer fora 99.999.989. I obtenir el nombre de primers trobats que són: 5.761.454.
7. L'execució del codi es farà variant N de 70M, 80M, 90M i 100M.
8. La idea és que cada thread faci un tros dels possibles primers fins a N. Cal vigilar on es guarden els primers trobats pels threads de manera que no se'n perdi cap.
9. El programa tindrà dos paràmetres: N i Nthreads.

```c
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#define N 100000000
#define INI 1500

int p[N/10];

int main(int na,char* arg[])
{
int i;
int nn,pp,num;

assert(na==2);    // nombre d'arguments
nn = atoi(arg[1]);

printf("Tots els primers fins a %d\n",nn);

p[0] = 2;
p[1] = 3;
pp = 2;
num = 5;

while (pp < INI)
 {
  for (i=1; p[i]*p[i] <= num ;i++)
    if (num % p[i] == 0) break;
  if (p[i]*p[i] > num) p[pp++]=num;
  num += 2;
 }

for(; num<nn; num+=2)
 {
  int div = 0; // No divisible
  for (i=1; p[i]*p[i] <= num && !div;i++)
    div = div || !(num % p[i]);
  if (!div) p[pp++]=num;
 }

printf("Hi ha %d primers\n",pp-1);
printf("Darrer primer trobat %d\n",p[pp-1]);
```