PRÀCTICA 1a: "ANÀLISI DE PRESTACIONS"

Arquitectura i Enginyeria dels Computadors (3r curs)
E.T.S. d'Enginyeria Informàtica (ETSINF)
Dpt. d'Informàtica de Sistemes i Computadors (DISCA)

Objectius:

- Aplicar la llei d'Amdahl.
- Avaluar i comparar les prestacions de distintes arquitectures.

Desenvolupament:

Llei d'Amdahl.

La llei d'Amdahl ens diu que si una aplicació fa servir un cert component durant una fracció F del seu temps d'execució, l'acceleració global que s'hi obtendria si es millorara en un factor S aquest component ve donada per l'expressió següent:

$$S' = \frac{1}{1 - F + \frac{F}{S}}$$

En moltes ocasions resulta dificil determinar la fracció F, bé per no disposar del codi font de les aplicacions, bé perquè l'ús del component està distribuït al llarg de tot el temps d'execució del programa. Per sort, l'aplicació "a la inversa" de la Llei d'Amdahl permet obtenir fàcilment aquesta fracció.

Per això, en necessita executar l'aplicació baix estudi amb dos tipus del component en questió la millara local S del qual siga coneguda. El quocient dels dos temps d'execució obtinguts és l'acceleració global S'. Coneguts S i S', només caldrà aïllar F.

Per a il·lustrar tot això amb un exemple, suposem que voleu obtenir la fracció F del temps d'execució que una aplicació, que fa operacions amb matrius (`matrix`), dedica a fer productes escalars. El producte escalar de dos vectors $A = \{a_1, a_2, ..., a_n\}$ i $B = \{b_1, b_2, ..., b_n\}$ d'n elements es defineix com

$$A \cdot B = \{a_1, a_2, ..., a_n\} \cdot \{b_1, b_2, ..., b_n\} = a_1b_1 + a_2b_2 + ... + a_nb_n$$

i és una operació bàsica, per exemple, en la multiplicació de matrius. La idea serà fer dues implementacions d'aquest component, una fent ús de les instruccions estàndard del processador i l'altra fent servir les instruccions del joc SSE (*Streaming SIMD Extensions*).

```
float Scalar(float *s1,
             float *s2,
             int size)
{

    int i;
    float prod = 0.0;

    for(i=0; i<size; i++) {
      prod += s1[i] * s2[i];
    }

    return prod;

} // end Scalar()
```

Figura 1: Producte escalar implementat amb instruccions estàndard.

```
float ScalarSSE(float *m1,
                float *m2,
                int size)
{
    float prod = 0.0;
    int i;
    __m128 X, Y, Z;

    Z = _mm_setzero_ps(); /* all to 0.0 */
    for(i=0; i<size; i+=4) {
      X = _mm_load_ps(&m1[i]);
      Y = _mm_load_ps(&m2[i]);
      X = _mm_mul_ps(X, Y);
      Z = _mm_add_ps(X, Z);
    }

    for(i=0; i<4; i++) {
      prod += Z[i];
    }

    return prod;

} // end ScalarSSE()
```

Figura 2: Producte escalar implementat amb instruccions SSE.

Càlcul de l'acceleració local *S*

Necessitem, per tant, disposar de les implementacions del component en qüestió. Les funcions ***Scalar()*** i ***ScalarSSE()*** implementen el producte escalar de dos vectors tal com s'ha comentat més amunt. La implementació amb instruccions SSE ha de ser més ràpida, ja que aquestes instruccions permeten operar simultàniament amb més d'una dada, en aquest cas fins 4 valors de coma flotant. En les figures 1 i 2 podeu veure les dues implementacions.

La relació entre la velocitat obtinguda per ambdues implementacions serà S en la expressió de la Llei d'Amdahl. Per a obtenir aquesta relació ens servirem d'un programa senzill en llenguatge C que fa productes escalars dins d'un bucle. La figura 3 mostra el codi del programa, nomenat `scalar.c`.

Noteu que en el codi hi ha dues línies amb el comentari:

```
\\ A MODIFICAR
```

Amb aquestes línies treballareu per a obtenir temps diferents d'execució i així estimar el valor de S. Haureu d'obtenir el temps d'execució per a ambdues implementacions (instruccions estàndard t_{std}, instruccions SSE t_{sse}) i així mateix el temps d'execució degut a la sobrecàrrega del bucle de mesura i la inicialització dels vectors (t_{load}). Restant aquest valor als anteriors podreu obtenir el temps real empleat per la component que ens interessa.

```
int main(int argc, char * argv[]) {

    int     i;
    time_t  start, stop;
    double  avg_time;
    double  cur_time;
    int     rep=10;
    int     msize=MSIZE;
    float   fvalue;
    char   *cmd;

    if (argc == 2) {
      rep = atoi(argv[1]);
    } else if (argc == 3) {
      rep = atoi(argv[1]);
      msize = atoi(argv[2]);
    } // end if/else
    fprintf(stderr, "Rep = %d / size = %d\n", rep, msize);

    for(i=0; i<rep; i++) {
      init_vector(vector_in, msize);
      init_vector(vector_in2, msize);
      fvalue = Scalar(vector_in2, vector_in, msize);       // A MODIFICAR
      //fvalue = ScalarSSE(vector_in2, vector_in, msize);  // A MODIFICAR
    } // end for

    exit(0);

} // end main()
```

Figura 3: Programa `scalar.c` utilitzat per a avaluar la relació de les dues implementacions del producte escalar.

Per a obtenir el temps t_{std} haureu de comprovar en el programa `scalar.c` que només la línia de la funció ***Scalar()*** està sense comentar,

```
    ...

    fvalue = Scalar(vector_in2, vector_in, msize);       // A MODIFICAR
    //fvalue = ScalarSSE(vector_in2, vector_in, msize);  // A MODIFICAR
    ...
```

Tot seguit heu de compilar el programa des d'una consola de linux,

```
gcc -O0 -msse -o scalar-std scalar.c
```

executar-lo i mesurar-ne el temps d'execució amb l'ordre `time`.

LES SEGÜENTS INSTRUCCIONS **NO** SÓN NECESÀRIES SI ES REALITZA LA PRACTICA EN UN ENTORN VIRTUAL (Polilabs, VirtualBox, VMware, ...)

No obstant, abans d'executar el programa heu de fixar la velocitat dels cores del processador perquè les mesures siguen uniformes. Per defecte, els cores estan en mode *ondemand* i la seua velocitat varia segons la càrrega, cosa poc adient per a prendre mesures. Per

a fixar la velocitat dels cores disposeu de l'ordre `cpufreq-set`, i per a confirmar l'ordre `cpufreq-info`. Fixareu la velocitat del processador a 3.3Ghz. El paràmetre `-c #` serveix per a indicar el core que anem a fixar,

```
cpufreq-set -c 0 -f 3.3GHz
cpufreq-set -c 1 -f 3.3GHz
cpufreq-set -c 2 -f 3.3GHz
cpufreq-set -c 3 -f 3.3GHz
```

el verificareu amb

```
cpufreq-info
```

o,

```
cat /proc/cpuinfo
```

i ja podeu procedir amb les mesures.

> **IMPORTANT: vos recordem que en linux el directori actual no està en el PATH, cosa que us obliga a col·locar *./* abans de l'ordre amb què voleu executar els vostres programes des de la consola. Per a estalviar-vos d'això, executeu en la consola on esteu treballant, o afegiu al final de l'arxiu $HOME/.bashrc, aquesta línia:**
>
> ```
> export PATH=./:$PATH
> ```

Com hem dit, executeu el programa `scalar-std` amb l'ordre

```
time scalar-std 100000 1024
```

on els paràmetres indiquen simplement que el bucle del programa es repetirà 100000 vegades i el llarg dels vectors serà de 1024 elements. Anoteu el temps empleat per la tasca (*user+system*) com t_{std}.

Ara modifiqueu de nou el programa `scalar.c` per a obtenir el temps t_{sse}. Ho fareu comentant la línia de la funció ***Scalar()*** i descomentant la línia de la funció ***ScalarSSE()***,

```
  ...
//fvalue = Scalar(vector_in2, vector_in, msize);   // A MODIFICAR
fvalue = ScalarSSE(vector_in2, vector_in, msize);  // A MODIFICAR
  ...
```

Compileu i executeu de nou el programa,

```
gcc -O0 -msse -o scalar-sse scalar.c
time scalar-sse 100000 1024
```

i anoteu-ne el temps d'execució com t_{sse}. Observeu que, en compilar, hem afegit la directiva ***-msse***, per a que el compilador genere codi utilitzant les instruccions SSE.

Finalment, editeu de nou el programa `scalar.c` i comenteu les línies que fan la crida al producte escalar, i així calculareu la sobrecàrrega deguda al bucle e inicializacions,

```
  ...
//fvalue = Scalar(vector_in2, vector_in, msize);   // A MODIFICAR
//fvalue = ScalarSSE(vector_in2, vector_in, msize);  // A MODIFICAR
  ...
```

compileu i executeu una vegada més el programa,

```
gcc -O0 -msse -o scalar-load scalar.c
time scalar-load 100000 1024
```

tot anotant-ne el temps d'execució com t_{load}. Amb aquesta informació ja podeu estimar el valor de S com

$$S = \frac{t_{std} - t_{load}}{t_{sse} - t_{load}}$$

Una vegada obtinguda la relació entre les dues implementacions (S), mesureu el temps d'execució de l'aplicació `matrix` sota estudi amb cadascuna de les implementacions. La relació entre els temps d'execució serà l'acceleració global que la aplicació assoleix en accelerar el producte escalar. Aquesta relació és S' en la Llei d'Amdahl. Per tant, podeu substituir S i S' en la llei d'Amdahl per tal d'aïllar F per a obtenir la fracció del temps que l'aplicació `matrix` dedica al producte escalar.

Càlcul de la fracció de temps local *F*

Heu d'obtenir la fracció de temps que l'aplicació `matrix` dedica a calcular productes escalars.

$$S' = \frac{t_{mat-std}}{t_{mat-sse}}$$

Per a obtenir $t_{mat-std}$ cal que editeu el codi font `matrix.c` i hi busqueu al principi la definició de la macro **__SCALAR_PROD()**,

```
  ...
#define __SCALAR_PROD(v1, v2, s)  Scalar(v1, v2, s);
//#define __SCALAR_PROD(v1, v2, s)  ScalarSSE(v1, v2, s);
//#define __SCALAR_PROD(v1, v2, s)
  ...
```

Deixeu sense comentar la versió estàndard tal como es mostra en el fragment de codi anterior. D'aquesta manera l'aplicació `matrix` farà servir aquesta implementació del producte escalar.
Compileu i executeu

```
gcc -O0 -msse -o matrix-std matrix.c -lm
time matrix-std 1 1024
```

En aquest cas feu només una execució. D'ací en endavant treballareu sempre amb matrius de 1024x1024, per tal que la grandària del vector siga sempre la mateixa. Anoteu el temps com $t_{mat-std}$.

Tot seguit editarem de nou el codi font `matrix.c` per seleccionar l'implementació de producte escalar amb instrucccions SSE,

```
  ...
//#define __SCALAR_PROD(v1, v2, s)  Scalar(v1, v2, s);
#define __SCALAR_PROD(v1, v2, s)  ScalarSSE(v1, v2, s);
//#define __SCALAR_PROD(v1, v2, s)
  ...
```

Compileu i executeu de nou,

```
gcc -O0 -msse -o matrix-sse matrix.c -lm
time matrix-sse 1 1024
```

i anoteu el temps com $t_{mat-sse}$.

$\Rightarrow$ Amb les dades obtingudes, calculeu el percentatge de temps F que l'aplicació dedica als productes escalars.

Càlcul experimental de la fracció de temps local (F_{exp})

Encara que en general no és possible, amb el codi de `matrix` i el component sota avaluació (el producte escalar), podeu calcular experimentalment la fracció del temps que l'aplicació dedica a fer productes escalars. Per a això, només heu de definir la macro del producte escalar com buida,

```
  ...
//#define __SCALAR_PROD(v1, v2, s)  Scalar(v1, v2, s);
//#define __SCALAR_PROD(v1, v2, s)  ScalarSSE(v1, v2, s);
#define __SCALAR_PROD(v1, v2, s)
  ...
```

i així podreu estimar quant de temps dedica el programa a la resta de tasques. A aquest temps el nomenarem $t_{mat-res}$ i ens permetrà deduir experimentalment la fracció del temps (F_{exp}) mitjançant la fòrmula següent

$$F_{exp} = \frac{t_{mat-std} - t_{mat-res}}{t_{mat-std}}$$

En compilar i executar la nova versió del programa,

```
gcc -O0 -msse -o matrix-res matrix.c -lm
time matrix-res 1 1024
```

obtindreu $t_{mat-res}$.

$\Rightarrow$ Obteniu experimentalment la F_{exp}, és a dir, la fracció del temps que `matrix` dedica al producte escalar. Compareu-la amb l'obtinguda mitjançant la llei d'Amdahl.

Anàlisi de les prestacions de les arquitectures.

En aquest apartat heu de mesurar les prestacions dels computadors del laboratori fent servir els programes següents:

- dos *benchmarks* sintètics (**dhrystone**, per a l'aritmètica d'enters, i **whetstone**, per a l'aritmètica de coma flotant)
- dues aplicacions reals: el compilador del llenguatge de programació C **gcc**, que només utilitza aritmètica entera, i l'aplicació **xv**, que fa processament d'imatges.

Per això, heu de mesurar el temps d'execució d'aquests programes, amb l'ajuda de l'ordre `time`:

- **dhrystone** (10.000.000 iteracions):

```
time dhrystone
```

  Indiqueu que ha de fer 10.000.000 iteracions.

  Anoteu el temps d'execució $T_{dhrystone}$ del programa.

- **whetstone** (10.000 iteracions):

  Ara teclegeu:

```
time whetstone 10000
```

  Anoteu el temps d'execució T_{whet-h} i d'altres dades que oferisca el programa.

- Compilador del llenguatge C, quan compila una aplicació:

  Compileu l'aplicació **xv**. Per això, tot suposant que els arxius font del programa estan ubicats dins de la subcarpeta `xv-310a/` del vostre directori de treball, teclegeu:

```
cd xv-310a
make clean
time make
```

  Anoteu el temps d'execució T_{gcc}.

- Aplicació *xv*

  Ara executem la applicació **xv** que acabem de compilar, obrint un fitxer de imatge:

```
cd prac1
time xv-310a/xv -wait 5 mundo.jpg
```

  Anoteu el temps d'execució T_{xv}.

La taula següent mostra els resultats obtinguts (temps d'execució en segons) en executar aquestes aplicacions en tres màquines distintes. La màquina C és el lloc de treball amb el què esteu treballant al laboratori:

Programa/Màquina	A	B	C
dhrystone	5	18	
whetstone	2.5	10	
gcc	40	130	
xv	4.5	15	

$\Rightarrow$ Compareu les prestacions dels tres computadors de tres formes diferents. La primera considerarà cadascun dels programes de manera aïllada, la segona farà la mitjana aritmètica dels temps d'execució, i la tercera la mitjana geomètrica dels temps d'execució normalitzats a la màquina B.