Arquitectura y Tecnología de Computadores. Curso 09/10

Arquitecturas Paralelas. **Práctica I**

EVALUACION DEL RENDIMIENTO DE COMPUTADORES (I)

Benchmarks Sintéticos

1. Objetivo

El objetivo de la presente práctica es familiarizarse con el empleo de los *benchmarks* utilizados para evaluar el rendimiento de los computadores. Para ello se utilizarán un par de *benchmarks* sintéticos escritos en C: el *benchmark* ***dhrystone***, que estima el rendimiento al trabajar con operaciones de CPU (datos enteros) y el *benchmark* ***whetstone***, que estima el rendimiento al trabajar con operaciones en punto flotante (datos reales). Estos *benchmarks* se utilizarán para evaluar las prestaciones de diferentes máquinas multiusuario en las que el alumno tiene cuenta.

2. Análisis a realizar

- Evaluar el rendimiento de los computadores disponibles utilizando *benchmarks* sintéticos
- Evaluar el rendimiento obtenido al ejecutar aplicaciones concretas
- Establecer comparativas de rendimiento basadas tanto en los *benchmarks* como en las aplicaciones
- Valorar el efecto de procesador y memoria sobre el rendimiento de CPU
- Valorar la precisión de la estimación de rendimiento proporcionada por los *benchmarks* sintéticos
- Valorar el efecto del compilador sobre el rendimiento de CPU

3. Pasos a seguir en la práctica

- Compilar y ejecutar los *benchmarks* en las máquinas disponibles, máquinas de diferentes fabricantes y con diferentes características. Anotar las mediciones resultantes.

- Ejecutar en cada una de las máquinas las aplicaciones, que operan bien sobre datos enteros o bien sobre datos flotantes. Las aplicaciones son (ver Anexo III):

gcc	el propio compilador de C de *gnu* (datos enteros)
mxvdouble	multiplicación de matriz por vector (datos flotantes en doble precisión)

  El programa *mxvdouble* ha de ser completado para poder medir su tiempo de ejecución en las mismas condiciones que se hace en los *benchmarks* utilizados. Para ello, antes de ser compilado (**con optimización**) habrá que añadir instrucciones de repetición de cálculos (para medir tiempo suficiente y mejorar así la precisión), toma de tiempos y visualización del tiempo de cálculo medido (las instrucciones aparecen en negrita en el listado del Anexo III). Una vez ejecutado el programa (**con matrices de 300x300**) devolverá entonces la medición de tiempo de CPU de usuario correspondiente.

  En el programa *gcc* los tiempos serán medidos invocando la propia compilación (**con optimización**) del programa *mxvdouble.* Además, con objeto de medir un tiempo significativo en todas las máquinas **se repetirá la compilación 10 veces** haciendo uso del script del Anexo III. Una vez creado el script se le dará permiso de ejecución mediante el comando *chmod +x nombre_script* (denominar al script por ejemplo “compila10”) y se obtendrá su tiempo de ejecución mediante el comando *time nombre_script.*

- Comparar las máquinas en base a las mediciones de los *benchmarks* utilizando la ganancia que supone una máquina respecto a otra como índice de la comparación ¿a que factores se deben fundamentalmente las diferencias de rendimiento obtenidas ? (tener en cuenta los factores que influyen sobre el tiempo de CPU de usuario que da lugar al rendimiento de procesador).

- Comparar ahora las máquinas en base a las mediciones de las aplicaciones utilizando de nuevo la ganancia como métrica. Valorar los resultados de ambas comparativas: ¿estiman adecuadamente los *benchmarks* el rendimiento obtenido con las aplicaciones? ¿son las estimaciones para enteros y flotantes igual de precisas? ¿a que crees que se debe todo ello? (tener en cuenta las diferencias existentes entre los *benchmarks* sintéticos y las aplicaciones consideradas).

- Comprobar experimentalmente el efecto del compilador sobre el tiempo de CPU trabajando **con y sin optimización** en el compilador y utilizando los compiladores nativos de algunas de las máquinas (*cc*) como alternativa al compilador *gcc* ¿tiene en general el compilador una influencia importante sobre el rendimiento de procesador?

En el Anexo I se indican las principales características de las máquinas disponibles. También se indican los ficheros y las ordenes de compilación necesarias para la obtención de los diferentes ejecutables en cada máquina. Mas información sobre las máquinas puede encontrarse en la página *web* de la asignatura.

4. Trabajo a entregar

Todos los *rankings*, tablas y demás tareas que se enumeran a continuación deben ser presentados en una hoja de calculo *EXCEL*, de acuerdo con el modelo del Anexo II. La hoja de cálculo, que se etiquetará como *Practica1* y se almacenará en un documento de nombre *"DNI_alumno.xls"*, será entregada en su momento al profesor como informe de la práctica a través de la *web* de la asignatura.

1. Confeccionar los *rankings* de rendimiento de las diferentes máquinas para cada uno de los *benchmarks* y aplicaciones (ordenadas por tanto de mayor a menor rendimiento).

2. Confeccionar dos tablas comparativas del rendimiento de todas las máquinas utilizadas, una para trabajo con datos enteros y otra para trabajo con datos flotantes.

   En cada tabla, referenciar cada una de sus cuadriculas mediante una abscisa y una ordenada, representando tanto en abscisas como en ordenadas los nombres de las máquinas. Anotar en cada cuadricula dos valores, correspondientes ambos a la ganancia de velocidad de la máquina "ordenada" (eje Y) con respecto a la máquina "abscisa" (eje X). Estas ganancias se calcularán respectivamente mediante:

   - comparación directa de las medidas de productividad obtenidas como resultado de la ejecución de los *benchmarks*
   - comparación inversa de los tiempos de ejecución medidos para las aplicaciones

3. Se trata en este punto de resumir los resultados obtenidos en el punto anterior y extraer un valor único indicativo de la precisión del *benchmark* a la hora de estimar el rendimiento que podemos esperar para nuestras aplicaciones. Para ello se calculará la diferencia media entre la estimación obtenida con el *benchmark* y la obtenida con la aplicación. A partir de las 12 parejas de ganancias ya calculadas, se puede obtener el valor medio buscado, en términos porcentuales, mediante la expresión:

$$\frac{\sum[(G_{benchmark}/G_{aplicación}) - 1]}{6} \times 100$$

   expresión solo aplicada a las 6 parejas de ganancias que generen valores positivos en el numerador de la misma, ya que las otras 6 parejas solo aportan información redundante.

   Este valor nos informa del grado de precisión de la estimación del *benchmark* para aplicaciones como las utilizadas, de tal modo que un porcentaje 0 significa precisión absoluta y valores mayores (que pueden ser perfectamente mayores que el 100%) significan precisión menor.

4. Compilar de nuevo el programa *mxvdouble* en las máquinas **eliminando la opción de optimización**. Calcular el incremento porcentual de velocidad conseguido con la optimización en cada máquina al ejecutar *mxvdouble*, comparando para ello el nuevo tiempo de ejecución con el original.

5. Volviendo a trabajar **sin optimización**, compilar de nuevo el programa ***mxvdouble*** utilizando un compilador nativo *(cc)* distinto al *gcc* en aquellas máquinas que dispongan de el, compilador específicamente diseñado para las arquitectura correspondiente (en las demás máquinas *cc* hará referencia al propio *gcc*). Calcular el incremento porcentual de velocidad en cada una de esas máquinas cuando se pasa de utilizar el compilador genérico *gcc* a utilizar el nativo *cc*, **comparando para ello el nuevo tiempo de ejecución de *mxvdouble* con el original**.

Anexo I: Máquinas disponibles (S.O. Unix)

Máquina	Lisa	Sirio	Orion	Centauro
Dirección	*lisa.epsig.uniovi.es*	*sirio.edv.uniovi.es*	*orion.edv.uniovi.es*	*centauro.aulario.uniovi.es*
IP	*156.35.141.8*	*156.35.151.2*	*156.35.151.5*	*156.35.114.81*
Modelo	Dell PowerEdge 2950	Sun Ultra 20 WorkStation	PC clónico	DEC AlphaServer 2100A 5/300
Año	2007	2005	2000	1996
S.O.	Red Hat 5 Linux (Unix)	Solaris 10 (Unix)	Ubuntu 7.04 Linux (Unix)	Digital UNIX 4.0E (Unix)
CPU	2xQuad Core Xeon X5355 (64 bit)	AMD Opteron 144 (64 bit)	Intel Pentium III (32 bit)	4x 21164 (64 bit)
Frecuencia	2,66 GHz.	1,8 GHz.	1 GHz.	291 MHz.
Cache	L1:128KB(I)+128KB(D) L2:8MB	L1:64KB(I)+64KB(D) L2:1MB	L1:16KB(I)+16KB(D) L2:256KB	L1:8KB(I)+8KB(D) L2:96KB
FPU	Integrada	Integrada	Integrada	Integrada
Memoria	16GB RAM	512MB RAM	256MB RAM	512MB RAM (Cache L3:2MB)
Compiladores	gcc	gcc y cc (nativo)	gcc	gcc y cc (nativo)

Benchmarks:

Lisa	/profesores/fran/4atc/dhrystone/*	/profesores/fran/4atc/whetstone/*
Sirio	/export/home/fran/4atc/dhrystone/*	/export/home/fran/4atc/whetstone/*
Orion	/home/fran/4atc/dhrystone/*	/home/fran/4atc/whetstone/*
Centauro	/profesores/fran/4atc/dhrystone/*	/profesores/fran/4atc/whetstone/*

Ordenes de compilación:

dhrystone	contenidas en el archivo Makefile; incluyen optimización y son ejecutadas con make (gmake en Sirio)
whetstone	contenidas en el archivo Makefile; incluyen optimización y son ejecutadas con make (gmake en Sirio)
mxvdouble	gcc –O -o mxvdouble mxvdouble.c (con optimización)
	gcc -o mxvdouble mxvdouble.c (sin optimización)

Anexo II: Modelo de hoja EXCEL

Mediciones:

	Lisa	Sirio	Orion	Centauro
dhrystone				
gcc				
whetstone				
mxvdouble				

1) *Rankings*

	Máquina	*VAX MIPS*
1		
2		
3		
4		

	Máquina	*gcc (s)*
1		
2		
3		
4		

	Máquina	*Whetstones/s*
1		
2		
3		
4		

	Máquina	*mxvdouble (s)*
1		
2		
3		
4		

2) Comparativa

Lisa	-			
	-			
Sirio		-		
		-		
Orion			-	
			-	
Centauro				-
				-
	Lisa	**Sirio**	**Orion**	**Centauro**

¿Estiman adecuadamente los *benchmarks* el rendimiento obtenido con las aplicaciones? Sí/No

3) Precisión

	Enteros	**Flotantes**
Diferencia media entre estimaciones de *benchmark* y aplicación (%)		

¿Son las estimaciones para enteros y flotantes igual de precisas? Sí/No

4) Compilador

	mxvdouble
Incremento de velocidad con optimización en Lisa (%)	
Incremento de velocidad con optimización en Sirio (%)	
Incremento de velocidad con optimización en Orion (%)	
Incremento de velocidad con optimización en Centauro (%)	

Incremento de velocidad al pasar de *gcc* a *cc* (y ejecutar *mxvdouble)* en Sirio (%)	
Incremento de velocidad al pasar de *gcc* a *cc* (y ejecutar *mxvdouble)* en Centauro (%)	

¿Tiene el compilador una influencia importante sobre el rendimiento de CPU?	Sí/No

Anexo III: Código fuente

Script para compilar n veces un programa

```
#!/bin/bash
# Compila n veces un programa
i=1
while [[ $i -le 10 ]]
do
  gcc –O -o programa programa.c
  let i=$i+1
done
```

Función dtime()

```
/*****************************************************/
/*  UNIX dtime(). This is the preferred UNIX timer.  */
/*  Provided by: Markku Kolkka, mk59200@cc.tut.fi    */
/*  HP-UX Addition by: Bo Thide', bt@irfu.se         */
/*****************************************************/
#include <sys/time.h>
#include <sys/resource.h>

struct rusage rusage;

double dtime()
{
   double q;

   getrusage(RUSAGE_SELF,&rusage);

   q = (double)(rusage.ru_utime.tv_sec);
   q = q + (double)(rusage.ru_utime.tv_usec) * 1.0e-06;

 return q;
}
```

mxvdouble.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define N   1000
#define M   1000

double X[N], A[M*N], Y[M];

main()
{
FILE *fiy;
int num,i,j,fil,col;
double starttime;
double endtime;
double dtime();
char *nomfi="y.dat";

printf("\n\n\nNumero de filas de la matriz:  ");
scanf("%d",&fil);
printf("\n\n\nNumero de columnas de la matriz:  ");
scanf("%d",&col);

fiy=fopen(nomfi,"wb");
printf("\n Inicializando la matriz....");
inicializa(X,A,fil,col,Y);

/* Toma del tiempo inicial justo antes de empezar el cálculo */
printf("\n Iniciando el calculo....");
starttime = dtime();

/* Multiplicacion*/
  for (num=0; num<1000; num++)
    for (i=0; i<fil; i++)
        for (j=0; j<col; j++)
            Y[i]= Y[i] + A[i*col+j]*X[j];

/* Toma del tiempo final justo después de finalizar el cálculo */
endtime = dtime();

/* Volcado a pantalla del tiempo medido con 2 decimales y formato long float (lf) */
printf("\n\n Tiempo total de calculo: %.2lf  sec \n",endtime-starttime);

/* Ahora escribe los resultados en ficheros separados */
/*    for (i=0; i< fil; i++)
        fwrite(&Y[i],sizeof(double),fil,fiy);
    fclose(fiy);
       putchar('\n');                                  */
}

inicializa(double X[N],double A[N*N],int fil,int col,int Y[M])
{
    int i,j,num;
    double pp;

    for (i=0;i< fil;Y[i++]=0);
    srand(24);

    for (i=0; i<fil; i++)
    {
        for (j=0; j<col; j++)
        {
             pp=rand();
             num=((rand()*10) > 5)?-1:1;
             A[i*col+j]= pp/(rand()*num);
        }
    }

    for (i=0; i<col; i++)
    {
         pp=rand();
         num=((rand()*10) > 5)?-1:1;
         X[i]= pp/(rand()*num);
    }
}
```