Análisis del rendimiento del computador basado en benchmarks

Área de Arquitectura y Tecnología de Computadores
Versión 1.0,
10/09/2016

Índice

Objetivos de la sesión

En esta sesión se introduce al alumno en los conceptos sobre medición del tiempo de ejecución de programas en GNU/Linux con el objeto de analizar el rendimiento del sistema. Para ello se utilizan las características que proporciona el sistema operativo para acceder a relojes de altas prestaciones del sistema. Además de medir el tiempo de ejecución de varios programas utilizados como benchmarks sintéticos, se comparará el rendimiento de varios computadores basándose en un benchmark de aplicación. Finalmente, se analizará el resultado obtenido comparándolo con los resultados de suites de bechmarks proporcionados por organizaciones como SPEC (*Standard Performance Evaluation Corporation*).

Conocimientos y materiales necesarios

Para obtener el máximo aprovechamiento de esta sesión, el alumno debe:

- Tener instalada y correctamente configurada la máquina virtual con el sistema operativo GNU/Linux modificado sobre el que se trabajará a lo largo del curso.

- Ser capaz de compilar y ejecutar programas escritos en C a través de una terminal del sistema operativo Linux, como se estudió en la sesión anterior.

- Durante la sesión se plantearán una serie de preguntas que puedes responder en el correspondiente cuestionario en el campus virtual. Puedes abrir el cuestionario en otra pestaña del navegador pinchando en el enlace mientras mantienes pulsada la tecla `Ctrl`.

1. Instrumentación del código fuente en GNU/Linux

Antes de iniciar el análisis de prestaciones de un computador es necesario configurar la máquina virtual para que se puedan tener resultados significativos en algunos apartados de la sesión. También, para que los resultados sean comparables, se debería mantener esa configuración a lo largo de toda la sesión.

- Antes de iniciar la máquina virtual, selecciónala en VirtualBox y pulsa sobre la opción Configuración del menú.

- Selecciona Sistema y pulsa sobre la pestaña *Procesador*.

- Incrementa el número de procesadores hasta 2.

- Acepta y arranca la máquina virtual.

- Cambia al directorio de tu repositorio y sincronízalo con el repositorio de Bitbucket para actualizar los cambios que se hubiesen realizado con anterioridad.

```
$> git pull
```

- Descarga los ficheros necesarios para la práctica y descomprímelos con estas órdenes:

```
$> wget http://rigel.atc.uniovi.es/grado/2ac/files/sesion1-2.tar.gz
```

-
```
$> tar xvfz sesion1-2.tar.gz
```

- Añade los ficheros al índice git y confirma los cambios con estas órdenes:

```
$> git add sesion1-2
```

-
```
$> git commit
```

- Cámbiate al directorio `sesion1-2`.

- Copia el fichero de la hoja de cálculo `tema1.xls` a Windows para poder modificarlo.

Puedes utilizar la versión portable del programa WinSCP para intercambiar ficheros entre tus máquinas Windows y Linux

entre tus máquinas Windows y Linux.

En ocasiones es necesario medir el tiempo de ejecución de un programa o de un fragmento de programa. Estas mediciones pueden utilizarse con múltiples fines, como comparar el rendimiento de varios sistemas o determinar cuál es la parte de un programa que más tiempo consume para tratar de optimizarla, por ejemplo.

Los sistemas operativos suelen proporcionar funciones de alto nivel que permiten el acceso a contadores de marcas de tiempo del procesador, o TSC (*Time Stamp Counter*). Estos contadores cuentan el número de pulsos de la señal de reloj del procesador. A partir del número de pulsos transcurridos entre dos eventos en el sistema, y conociendo la frecuencia de dicha señal de reloj, se puede determinar con una precisión elevada el tiempo transcurrido entre ambos eventos. Diferentes sistemas operativos proporcionan diferentes métodos para acceder a estos contadores.

La API POSIX que implementa Linux proporciona para los lenguajes C y C++ la siguiente función que permite obtener una marca de tiempo de alguno de los relojes utilizados por el sistema, como el reloj de tiempo real (`CLOCK_REALTIME`).

```
int clock_gettime(clockid_t clk_id, struct timespec * tp);
```

Obtiene el tiempo del reloj especificado por `clk_id` y lo almacena en la variable apuntada por `tp`. Retorna 0 en caso de que la función se ejecute correctamente y -1 en caso contrario (en este caso la variable `errno` toma el valor apropiado para identificar el error).
La variable de tipo `timespec` sobre la que se almacena la marca de tiempo es una estructura que contiene la siguiente información, donde `time_t` es un dato de tipo entero que depende de la configuración del sistema.

```
struct timespec {
    time_t   tv_sec;        /* seconds */
    long     tv_nsec;       /* nanoseconds */
};
```

Esta función se puede utilizar para medir con precisión el tiempo transcurrido entre dos eventos del sistema, como muestra el siguiente esqueleto de programa.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

```c
#include <time.h>

int main()
{
    struct timespec tStart, tEnd;
    double dElapsedTimeS;

    // Start time measurement
    clock_gettime(CLOCK_REALTIME, &tStart);

    //  Code to be measured
    ......
    ......

    // Finish time measurement
    clock_gettime(CLOCK_REALTIME, &tEnd);

    //  Compute the elapsed time, in seconds
    dElapsedTimeS = (tEnd.tv_sec - tStart.tv_sec);
    dElapsedTimeS += (tEnd.tv_nsec - tStart.tv_nsec) / 1e+9;

    return 0;
}
```

Fíjate que al calcular el tiempo transcurrido en segundos se hace una conversión de entero a real de precisión doble que podría ocasionar pérdidas de precisión. Para los valores que se van a medir en esta práctica, ni muy pequeños ni muy grandes, esto no es importante, pero podría serlo en otro contexto.

2. Características del sistema a medir

Antes de comparar el rendimiento entre varios computadores vamos a identificar las características de cada uno de ellos. Esta información se puede obtener bien a través de la información que proporciona el sistema operativo o cualquier otro programa que se ejecute sobre el mismo. A continuación se te propondrán una serie de comandos y utilidades con los que podrás obtener la información necesaria. Alguno de ellos requerirá la instalación previa en el sistema operativo. También será necesario, para obtener más información, ejecutarlos como *root* usando el comando

`sudo`.

Debes apuntar la información que vas a obtener en la primera hoja de la hoja de cálculo de resultados `tema1.xls`. Esta hoja está cubierta con la información del equipo que se utiliza como base, S_{ref}, o sistema de referencia, para comparar el rendimiento de otro equipo, *SUT* (*System Under Test*). El equipo *SUT* será la máquina virtual de prácticas sobre la que estás desarrollando esta sesión.

- Empieza obteniendo información del procesador del sistema con los comandos `lscpu` y `lshw`. También puedes consultar el fichero de información del sistema.

```
$> cat /proc/cpuinfo
```

Recuerda que la ayuda se obtiene pasándole al comando la opción `-h` y que la salida de un comando *UNIX* se puede redirigir a un archivo con el operador `>`. También puedes buscar por Internet a partir del modelo de procesador que te muestre el sistema.

- El modelo de la placa base puedes obtenerlo con el siguiente comando. El resto de información tendrás que buscarla en Internet a partir del modelo.

```
$> sudo dmidecode -t 2
```

- Obtén ahora información de la memoria que se te solicita utilizando alguna de estos comandos: `free`, `top` o `lshw`. También puedes consultar el fichero de información del sistema.

```
$> cat /proc/meminfo
```

- El tipo de memoria puede consultarse con el siguiente comando

comando:

```
$> sudo dmidecode -t 17
```

Es muy problable que al ejecutar Linux en máquina virtual este comando no te devuelva información útil. En ese caso, deja en blanco la casilla correspondiente en la hoja excel.

- Otros comandos que podrías utilizar son `hwinfo` y `inxi`, si bien tendrías que instalarlos previamente.

Puedes instalar ambos paquetes con el siguiente comando `sudo apt-get install hwinfo inxi`.

- Intenta obtener, mediante la información que te ofrece el sistema operativo, el nombre del disco en el que está instalado y apúntalo en la hoja de cálculo.

- Completa la información de la hoja de cálculo con la versión del sistema operativo sobre el que estás desarrollando esta sesión de prácticas. Puedes obtener esta información a través del comando del sistema.

```
$> cat /etc/lsb-release
```

Ten en cuenta que vamos a realizar las pruebas de rendimiento sobre la máquina virtual y no sobre una máquina física, por lo que la capa de virtualización añadirá cierta sobrecarga.

3. Análisis del rendimiento mediante carga sintética

monolítica

Los benchmarks sintéticos, o programas para medir el rendimiento de un computador basados en carga sintética, son programas que intentan reproducir las operaciones más habituales que se desarrollan en los programas reales. Dependiendo de la carga que se quiera reproducir, estos programas pueden ejecutar compiladores o traductores de lenguajes de programación, cálculos matemáticos complejos, procesamiento de gráficos, etc.

En este apartado vamos a trabajar con benchmarks sintéticos que realizan operaciones sobre números reales con estructuras de datos de diferentes tamaños, lo que, además del procesador del sistema, involucra al sistema de memoria y al sistema de entrada/salida del computador. Estos benchmarks podrían ser útiles para comparar el rendimiento de computadores orientados a realizar cálculos científicos, por ejemplo.

El fichero `bench-fp` contiene el código incompleto de un programa que accede al reloj de altas prestaciones del sistema para medir el tiempo de ejecución de la función `Task`. Esta función se encarga de invocar a la función `DoFloatingPoint` con diferentes parámetros. La función `DoFloatingPoint` reserva espacio en memoria para tres vectores de números reales (mediante la función `malloc`), inicializa dos de ellos con valores aleatorios (mediante la función `rand`) y realiza varias operaciones aritméticas sobre números reales para inicializar los elementos del tercer vector.

A continuación se muestra una breve descripción de las funciones que aparecen en el programa:

```
void * malloc(size_t size);
```

Reserva tantos bytes en memoria como los indicados por `size` y devuelve un puntero a la memoria reservada. La zona de memoria reservada no se inicializa con ningún valor. En caso de que la función no se ejecute correctamente el valor retornado es `NULL`.

```
int rand(void);
```

Retorna un valor entero pseudo-aleatorio en el rango [`0`,`RAND_MAX`].

```
void srand(unsigned int seed);
```

Coloca el valor `seed` como semilla para la generación de números pseudo-aleatorios que devolverá la función `rand`. Las secuencias de números retornadas por `rand` serán idénticas cuando se invoque a `srand` con el mismo valor de la semilla.

- A la vista del código, ¿cuántas veces se ejecuta la instrucción `pdDest[i] = sqrt(pdDest[i])` cada vez que se ejecuta el programa? Responde en el cuestionario: pregunta 1.

- Completa el código del programa en los puntos en los que aparece `<<Complete>>` de acuerdo al esquema mostrado en la primera sección del enunciado de esta sesión.

- Accede, a través de una terminal del sistema operativo, a la carpeta `bench_fp`. Compila y enlaza el programa utilizando el `Makefile` que se acompaña.

- Ejecuta el programa bench_fp. Recuerda que debes indicar la ruta relativa.

  ```
  $> ./bench_fp
  ```

  Si has completado correctamente el fichero fuente, se debería mostrar en la consola el tiempo transcurrido durante la ejecución de la función `Task`.

- Dado que una única medición puede contener un error

que no somos capaces de estimar, ejecuta el programa otras cuatro veces. Anota los cinco tiempos medidos en la hoja de cálculo proporcionada para anotar los resultados de la práctica (dentro de la hoja `1-2`, en las columnas `t_1` a `t_5` de la fila `bench_fp`), calcula el valor medio de estos tiempos de ejecución y su desviación típica mediante sendas fórmulas en las columnas `Media t_1 a t_5` y `Desv. Típ. t_1 a t_5`, y el intervalo de confianza para la media con un nivel de confianza del 95% también con una fórmula.

- Si se considera a la función `Task` como la *tarea* a ejecutar por el sistema, ¿cuál es la productividad media del sistema expresada en tareas por minuto? Anota el resultado en la hoja de cálculo en la columna `Productividad (tareas/min)`.
  ¿Cuál es la fórmula que has utilizado en esa celda para realizar el cálculo? Responde en el cuestionario: pregunta 2.

- Calcula la aceleración del *SUT* con respecto al sistema de referencia utilizando el programa `bench_fp` como carga. Anota el resultado en la hoja de cálculo.
  ¿Cuál es la fórmula que has utilizado en esa celda para realizar el cálculo? Responde en el cuestionario: pregunta 3.

- A la vista del programa, y de las características hardware del procesador que has obtenido anteriormente, ¿crees que este benchmark es realmente útil para medir las

capacidades del sistema en cuanto a cálculo científico? Para comprobar tu respuesta, ejecuta en otra ventana el comando `top` del sistema. El programa `top` muestra la información del estado del sistema cada `n` segundos. En la parte de arriba, muestra un resumen del sistema y, en las filas inferiores, muestra el detalle proceso a proceso. Si se desea información más frecuentemente, se puede modificar el intervalo de captura de `top` con la siguiente opción:

```
$> top -d ss.t
```

donde *ss* son los segundos y *t* las décimas de segundo. Por defecto, `top` captura cada 1.5 segundos.

¿Cuál es el porcentaje de uso del procesador que muestra `top` en la cabecera? Responde en el cuestionario: pregunta 4.

¿Cuál es el porcentaje de uso del procesador que muestra `top` en la línea específica de `bench-fp`? Responde en el cuestionario: pregunta 5.

¿Sabrías explicar la razón de las discrepancias? Si tienes dudas pregúntale a tu profesor.

4. Análisis del rendimiento mediante carga sintética multihilo

Independientemente de otros factores que pueden afectar, como el sistema de memoria, el rendimiento del programa `bench_fp` está limitado por el rendimiento

del sistema en la ejecución de un único hilo al ser un programa monohilo. En los sistemas actuales, en los que los procesadores suelen tener varios núcleos (cada encapsulado contiene realmente varios procesadores físicos ensamblados de forma conjunta) y en los que cada núcleo puede dar soporte a la ejecución de varios hilos de forma simultánea (como la tecnología HyperThreading de Intel), para obtener el máximo aprovechamiento del sistema es necesario recurrir a la programación paralela (o ejecutar varias instancias del mismo programa monohilo).

A continuación vamos a estudiar conceptos básicos sobre hilos para poder mejorar nuestro benchmark. Un hilo es la unidad mínima planificable por el sistema operativo. Por defecto, un programa tiene un único hilo de ejecución, por lo que la creación y gestión de hilos dentro de un programa es responsabilidad del programador, o de la biblioteca de hilos que utilice. En este último caso, el estándar POSIX define la API para la creación y gestión de hilos dentro de una aplicación.

Para crear un hilo se debe utilizar la función siguiente, declarada en el fichero de cabecera `pthread.h`.

```
int pthread_create( pthread_t * thread, const pthread_attr_t *
attr,
     void * (*start_routine) (void *), void * arg);
```

Crea un hilo y lo pone listo para ejecutar. En el parámetro `thread` se almacena el identificador del hilo creado, mientras que a través del parámetro `attr` se pueden indicar atributos de creación del hilo, aunque normalmente se le pasará `NULL`. El parámetro `start_routine` es un puntero a la función que ejecutará el hilo, que recibirá como paramétro el indicado en `arg`. Si la función tiene éxito retorna 0 y un código de error en caso contrario.
El hilo principal, o cualquier otro hilo, puede esperar a que finalice un hilo para continuar. Para ello, debe utilizar la siguiente función.

```
int pthread_join(pthread_t thread, void ** retval);
```

Espera por la finalización de un hilo. El hilo por el que se espera se indica a través del parámetro `thread`. Se puede recoger el puntero retornado por el hilo al finalizar en el parámetro `retval` o indicar `NULL` para descartarlo. De cualquier forma, ten en cuenta que el puntero retornado por el hilo debe apuntar a una zona de memoria

válida aun después de finalizado el hilo, por lo que habitualmente se retornarán punteros apuntando a una zona del montículo (memoria reservada con `malloc`). Si la función tiene éxito retorna 0 y un código de error en caso contrario.
A continuación se muestra un ejemplo muy simple de programa multihilo.

Fichero 1-2thread.c

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
```

```
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
```

```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

void* ThreadProc(void* arg)
{
    const int TIMES = 6;
    int i;

    // Cast
    int n = *((int*)arg);
    for (i = 0; i < TIMES; i++)
    {
        printf("Thread %d, message %d\n", n, i);
        sleep(1);  // Sleep 1 second
    }
    printf("Thread %d finished.\n", n);
    return NULL;
}

int main(int argc, char* argv[])
{
    const int N = 5;
    const int TIMES = 3;
    pthread_t thread[N];
    int i;
    int a[N];
```

```
    // Thread creation
    for (i = 0; i < N; i++)
    {
        a[i]=i;
        if (pthread_create(&thread[i], NULL, ThreadProc, &a[i]) != 0)
        {
            fprintf(stderr, "ERROR: Creating thread %d\n", i);
            return EXIT_FAILURE;
        }
    }

    for (i = 0; i < TIMES; i++)
    {
        printf("Main thread, message %d\n", i);
        sleep(1);  // Sleep 1 second
    }

    // Wait till the completion of all threads
    printf("Main thread waiting...\n");
    for (i = 0; i < N; i++)
    {
        pthread_join(thread[i], NULL);
    }
    printf("Main thread finished.\n");
    return EXIT_SUCCESS;
}
```

- Crea un `Makefile` para que el programa `1-2thread.c` se compile y enlace mediante la siguiente orden.

```
$> gcc 1-2thread.c -o 1-2thread -lpthread
```

- Ejecuta el programa varias veces e intenta comprender qué pasa con la salida por pantalla. ¿Puedes explicarlo?

Una vez que hemos estudiado cómo escribir un programa multihilo en C, en la carpeta `bench-fp-mt` se encuentra una versión multihilo creada a partir del programa `bench_fp`. Puedes observar que el programa declara una constante denominada `NUM_THREADS` que se inicializa con el número de hilos que se pretenden ejecutar en el programa (más un hilo que ejecuta la función `main`). Cada uno de estos hilos se crea mediante la función `pthread_create`, que recibe, entre

otros parámetros, la dirección de la función que ejecutará el hilo creado. En este caso todos los hilos ejecutarán la función `Task`. A continuación, lo que hace el hilo principal, el que ejecuta la función `main`, es esperar a que los `NUM_THREADS` hilos creados anteriormente finalicen su ejecución antes de finalizar la ejecución de la función `main`. Esta sincronización es necesaria dado que todo programa en C o C++ finaliza cuando se acaba de ejecutar la función `main`. Por lo tanto, sin esta sincronización el programa finalizaría sin que hubieran finalizado los hilos que se crean para ejecutar la función `Task`. Dicha sincronización se realiza mediante la función `pthread_join`.

- A la vista del código, ¿cuántas veces se ejecuta la instrucción `pdDest[i] = sqrt(pdDest[i])` cada vez que se ejecuta el programa? Responde en el cuestionario: pregunta 6.

- Completa el código del programa en los puntos en los que aparece `<<Complete>>` de acuerdo al esquema mostrado en la primera sección del enunciado de esta sesión.

- Compila, enlaza y ejecuta el programa.

- Al igual que con el programa monohilo, ejecuta otras cuatro veces este programa y anota las cinco mediciones de tiempo en la hoja de cálculo. Calcula además el tiempo promedio de la ejecución del programa y su desviación típica mediante sendas fórmulas en las columnas `Media t_1 a t_5` y `Desv. Típ. t_1 a t_5` respectivamente, y el intervalo de confianza para la media con un nivel de confianza del 95% también con una fórmula.

- Si se considera a la función `Task` como la *tarea* a ejecutar por el sistema, ¿cuál es la productividad media del sistema expresada en tareas por minuto? Anota el resultado en la hoja de cálculo en la columna `Productividad (tareas/min)`.
  ¿Cuál es la fórmula que has utilizado en esa celda para realizar el cálculo? Responde en el cuestionario: pregunta 7.

  ¿Es el número de veces que se incrementa la productividad superior, inferior o igual al incremento en el número de núcleos del procesador empleado? Responde en el cuestionario: pregunta 8.

- Calcula la aceleración del *SUT* con respecto al sistema de referencia utilizando el programa `bench_fp_mt` como carga. Anota el resultado en la hoja de cálculo.
  ¿Cuál es la fórmula que has utilizado en esa celda para realizar el cálculo? Responde en el cuestionario: pregunta 9.

- Ejecuta una vez más el programa observando a la vez los resultados que muestra `top`.
  ¿Cuál es el porcentaje de uso del procesador que muestra `top` en la cabecera? Responde en el cuestionario: pregunta 10.

  ¿Cuál es el porcentaje de uso del procesador que muestra `top` en la línea específica de `bench-fp_mt`? Responde en el cuestionario: pregunta 11.

en el cuestionario: pregunta 11.

¿Sabrías explicar la razón de las discrepancias?

5. Análisis del rendimiento mediante carga real

Los benchmarks basados en carga real o benchmarks de aplicación consisten en la ejecución de aplicaciones reales para comparar el rendimiento entre diferentes sistemas. Estas aplicaciones deberían representar la carga final que va soportar el sistema para que las comparaciones sean útiles.

En este apartado vamos a trabajar con una aplicación real que se utilizará para comparar el rendimiento entre dos sistemas. La aplicación se llama *Mencoder* y se utiliza para codificar archivos de vídeo.

- Inicia el programa `top`.

- Accede a la carpeta `mencoder` a través de otra terminal del sistema operativo. Ejecuta el script `x264` indicándole al sistema operativo que muestre el tiempo de ejecución del mismo de la siguiente forma.

```
$> time ./x264
```

  Ten en cuenta que, para poder ejecutarlo, debes proporcionarle al *script* permisos de ejecución. Tal como se ha visto en la sesión anterior, puedes hacerlo así:

```
$> chmod +x x264
```

Este *script* invoca el programa Mencoder para que reescale un vídeo utilizando el codec x264. El *script* indica en las opciones de invocación de Mencoder que el codec utilice un solo hilo, por lo que `top` debería mostar que solo una CPU tiene un rendimiento del 100%.

- 

- Puedes ver el contenido del

- Por defecto, `top` muestra el uso de CPU para una única CPU, por lo que podrían aparecer porcentajes por encima del 100%. Debes pulsar `1` mientras `top` está corriendo para mostrar el uso medio de cada CPU.

script siguiente comando

o .

```
$ >more x264
```

- Ejecuta el *script* cinco veces y obtén las estadísticas del tiempo de respuesta de la aplicación. Calcula además la productividad, es decir, cuántos vídeos por minuto es capaz de codificar el sistema, y la aceleración del *SUT* con respecto al sistema de referencia.
  ¿Cuál es la fórmula que has utilizado para calcular la productividad? Responde en el cuestionario: pregunta 12.

La implementación del codec x264 es multihilo, por lo que vamos a realizar la misma tarea de reescalar el vídeo, pero en este caso utilizando tantos hilos como procesadores tenga el sistema, con lo que se reparte la tarea entre varios hilos.

- Ejecuta desde la terminal del sistema operativo el siguiente comando en el directorio `mencoder`:

```
$> time ./x264_mt
```

  Observa que todos los procesadores del sistema deben estar en uso. Ejecuta el *script* cinco veces y obtén las estadísticas del tiempo de respuesta medio de la aplicación. Calcula además la productividad del *SUT* y su aceleración con respecto al de referencia.

  ¿Cuál es la fórmula que has utilizado en este caso para calcular la productividad? Responde en el cuestionario: pregunta 13.

- En la página web de ayuda de Mencoder se afirma, en relación con la implementación multihilo del codec x264, que

- *[…] increase encoding speed linearly with the number of CPU cores (about 94% per CPU core), with very little quality reduction […]*

  — Mencoder Docs

- ¿Se cumple esta afirmación en las mediciones que has tomado?

Una vez que tienes en la hoja de cálculo los valores de aceleración del *SUT* en comparación con el sistema de referencia para los programas `bench_fp`,

`bench_fp_mt` y `mencoder` en Linux, podemos calcular un ratio de aceleración del *SUT* con respecto al S_{ref} teniendo en cuenta todos estos resultados. Aunque podríamos pensar en la media aritmética del valor de aceleración de todas las aplicaciones, en casos en los que se comparan números normalizados, como ratios de rendimiento, suele ser más apropiado el uso de la media geométrica, que, dados n valores $x_1, x_2, \dots, x_n$, se calcula como se muestra a continuación:

$$ G = \sqrt[n]{x_1 \cdot x_2 \cdot \dots \cdot x_n} = \left (\prod_{i=1}^n x_i \right) ^{\frac{1}{n}} $$

- Calcula el ratio de aceleración del *SUT* en relación al sistema de referencia utilizando la media geométrica y anota el resultado en la celda `C17` de la hoja de cálculo.

6. Análisis del rendimiento mediante suites de benchmarks

Debido a la necesidad de comparar el rendimiento de sistemas en muchos entornos, científicos, industriales, académicos, etc., se han desarrollado colecciones de programas estandarizados con este fin. Estas colecciones de programas, denominadas en inglés *benchmark suites*, tratan de ser tests representativos de la carga final de determinados tipos de aplicaciones en el sistema. Los *benchmark suites* más conocidos en la actualidad están definidos y mantenidos por la organización SPEC (*System Performance Evaluation Corporation*).

De entre todas las suites de benchmarks desarrolladas por SPEC, la más conocida y la más comúnmente utilizada es SPEC CPU (su versión más reciente data de 2006: SPEC CPU2006), utilizada para medir el rendimiento de aplicaciones con carga intensiva de procesador y carga baja en entrada/salida. En esta colección de programas se encuentran aplicaciones reales que han sido migradas a la mayoría de plataformas actuales. Consiste en 12 programas que realizan operaciones sobre números enteros (SPECint2006 o CINT2006), escritas en C y C`, y 17`

`programas que realizan operaciones de punto flotante (SPECfp2006 o CFP2006), escritas en C, C` y Fortran.

Tanto CINT2006 como CFP2006 proporcionan dos tipos de métricas: SPECspeed y SPECrate. La primera mide tiempos de respuesta y la segunda productividad. Además, cada una de estas dos métricas proporciona dos valores: *base* y *peak*. El primero se obtiene de la media geométrica de los resultados proporcionados (bien por los 12 o los 17 tests, según sea el caso de enteros o reales) cuando los programas se compilan con optimizaciones normales, y el segundo (*peak*) se obtiene de la media geométrica de los resultados proporcionados por los tests cuando estos se compilan con optimizaciones para maximizar el rendimiento. Por lo tanto, los resultados proporcionados por SPEC CPU2006 para cada sistema serán los siguientes:

- SPECint2006:
    - CINT2006:
        - SPECint2006 (*peak*)
        - SPECint_base2006
    - CINT2006 Rates:
        - SPECint_rate2006 (*peak*)
        - SPECint_rate_base2006
-

- SPECfp2006:
    - CFP2006:
        - SPECfp2006 (*peak*)
        - SPECfp_base2006
    -
    - CFP2006 Rates:
        - SPECfp_rate2006 (*peak*)
        - SPECfp_rate_base2006
    -
-

La suite SPEC CPU2006 utiliza como sistema de referencia una Sun Ultra Enterprise 2 (procesador UltraSPARC II a 296 MHz), en la que, tanto los ratios proporcionados por SPECint2006 como por SPECfp2006 tienen el valor 1.0. Un sistema con un ratio SPECint2006 de 2.0 es el doble de rápido, en términos de cálculos sobre números enteros, que una Sun Ultra Enterprise 2.

- Accede a la página web de SPEC y busca los resultados publicados para la suite SPEC CPU2006.

- Trata de buscar los resultados de un sistema con un procesador similar al del equipo que estás utilizando para realizar esta sesión de prácticas (en la web se proporciona un formulario de búsqueda para filtrar los resultados por componentes hardware del sistema, como procesador, placa base, etc.).

- Una vez localizado un equipo similar, puedes ver los resultados de cada prueba (CINT2006, CFP2006, CINT2006 rates y CFP2006 rates) en múltiples formatos: HTML, CSV, PDF, etc. Elige, por ejemplo, el PDF en CINT2006. Observa cómo se muestran los resultados. Para cada test, bien de enteros o de flotantes, se muestra la aceleración del *SUT* con respecto al sistema de referencia. Puedes comprobar que el ratio final, por ejemplo, SPECint_base2006, es la media geométrica de las aceleraciones obtenidas en cada test de enteros.

- Anota los ratios para el *SUT* que se piden en la hoja de cálculo. ¿Son comparables los ratios de aceleración del *SUT* que has obtenido en los benchmark sintéticos y de aplicación anteriores con los proporcionados por SPEC?

Archivos de la práctica

Añade los archivos `bench_fp.c` y `bench_fp_mt.c` completados y la hoja de cálculo `tema1.xls` a tu repositorio *git* local y realiza una copia en tu repositorio de

Bitbucket. Recuerda los pasos a realizar.

- Añadir los ficheros o los cambios que hayamos realizado que queremos que estén bajo el control de versiones (de los que queremos *backup*) con la siguiente orden, donde `<fichero>` es el nombre del fichero añadido o modificado.

```
$> git add <fichero>
```

- Confirmar los cambios realizados, de forma que se registren en el sistema de control de versiones. Se solicitará un mensaje en el que indicarás una breve descripción de los cambios realizados.

```
$> git commit
```

- Por último, enviar los cambios hacia el repositorio de Bitbucket.

```
$> git push
```

Ejercicios adicionales

Se proponen los siguientes ejercicios adicionales:

- Modifica la configuración de la máquina virtual y reduce el número de procesadores de 2 a 1. Repite las medidas realizadas con los programas

`bench-fp` y `bench-fp-mt`. ¿Se obtienen los mismos resultados? ¿Cuál de los dos bechmarks produce más discrepancias con lo ya calculado? ¿Cuál crees que es la razón?

- Sería altamente recomendable que leyeras la descripción que hace SPEC de la suite de benchmarks CPU2006, dado que es uno de los benchmarks de referencia actualmente. Puedes acceder a esta descripción a través del siguiente enlace: http://www.spec.org/cpu2006/Docs/readme1st.html.