ASD: ARQUITECTURA DE SISTEMAS DISTRIBUIDOS. 3º GTI.
PRÁCTICA 3C: PROCESADORES SUPERESCALARES: LIMITES DEL ILP.
TRABAJO EN GRUPO

OBJETIVOS.

En esta práctica los alumnos estudiarán los límites de ILP que se pueden conseguir con técnicas dinámicas en GPP. Los alumnos repartidos en distintos grupos, observarán diferentes casos en los que prácticamente se alcanza el límite del flujo de datos (*data-flow limit*), de manera que el tiempo medio por iteración de un pequeño bucle es debido a tal límite. Los alumnos tendrán que exponer los resultados de esta práctica en el laboratorio (ver último apartado PRESENTACIONES DE LOS RESULTADOS DE LA PRÁCTICA).

PREPARACIÓN.

A cada uno de los grupos (de 2 o 3 alumnos) se le asignará un problema a resolver. Deberán ejecutar y analizar los resultados tanto del código que se le haya asignado (escribir en la función *problem*) como del código de ejemplo suministrado (función *example*). Hay un proyecto de Visual Studio en Enseñanza Virtual (ev.us.es) (sirve de plantilla para medir tiempos y probar las funciones *problema, example*).

REALIZACIÓN DE LA PRÁCTICA.

Cada grupo deberá resolver el problema asignado y descubrir si su problema es totalmente "paralelizable" y no tiene saltos condicionales dentro del bucle ni instrucciones que puedan bloquear la CPU. Deberá también analizar el comportamiento del código proporcionado como ejemplo *example()* , y explicar las diferencias obtenidas.
Se deben recoger como mínimo los resultados siguientes:

- Máquina (sobre todo procesador) donde se ejecuta el código. Se aconseja usar CPU-Z (descargar de http://www.cpuid.com/softwares/cpu-z.html), el cual indica muchas de las características del procesador.
- Compilador y Opciones de optimización usadas (sólo si algún grupo usa otro que no sea el del proyecto suministrado).
- Códigos de C usados en los bucles.
- Códigos de Ensamblador del x86 que crean interesantes para explicar los resultados de tiempo.
- Duración en ciclos por iteración (del for) de todos los bucles
- Realizar pruebas con tamaños mayores de vectores (cambiar valor de la constante N_ITER)
- Intente calcular el flujo de datos y encontrar el camino crítico y la UF (unidad funcional) de cada prueba. Recuerde los 5 tipos de bloqueo cuando se alcanza algún límite del ILP (última transparencia "Casos de estudio" de ASD tema 2 Técnicas dinámicas TFG.pdf)
- Explicar la causa de la duración y las variaciones (si las hay) que se encuentran al cambiar de código y al aumentar N_ITER (estudiar el efecto de los accesos a caché y a RAM).
- Y por supuesto cualquier otro resultado que considere importante.

Con todo lo anterior, se deberá redactar una presentación (estilo Power Point u OpenOffice Impress) de aprox. **8 minutos de duración, dividida en 2 partes aprox. iguales**.

Opciones a implementar para la función *problem()*.
Para cada problema, el bucle debe incluir al menos (**el alumno puede ir probando otras variantes para demostrar a qué se deben los tiempos que se obtienen, o para analizar nuevos resultados**):

1) Incluir divisiones donde el denominador tenga algún elemento de vectores (no sea una constante). Probar al menos con:
   a. Una sola división por cada iteración
   b. Dos divisiones por cada iteración

   ¿Por qué se pide que el denominador no sea constante?

2) Provocar muchos accesos a memoria por iteración, sin que haya operaciones de tipo acumulador (sumatorio, producto múltiple, etc.), ni tampoco dependencias entre iteraciones (recurrencia). Probar al menos con:
   a. 4 accesos por iteración
   b. 8 accesos por iteración
   c. 12 accesos por iteración

   Calcular el ancho de banda por iteración en cada caso. ¿Cuántos accesos se necesitan por iteración para saturar el ancho de banda? (necesitará probar con más casos además de los que se muestran como ejemplo). Recuerde que se han definido varios vectores en el proyecto de VS para facilitar esta tarea (*a[], b[]… h[]*).

3) Debe tener una sentencia *if()*. En el cuerpo del *if()* hay que realizar algún tipo de operación aritmética con vectores (sin que haya operaciones de tipo acumulador -sumatorio, producto múltiple, etc.-, ni tampoco dependencias entre iteraciones –recurrencia-). Probar al menos con tres tipos de condiciones del *if()*:
   a. La condición sea difícilmente predecible (p. ej. Random). Cuidado: no introducir la llamada a *función rand()* (o similar) en el cuerpo del bucle (tarda mucho).
   b. La condición sea fácilmente predecible por la BTB.
   c. En la condición del *if()* se incluya un cálculo con uno o más elementos de los vectores *cond* (sumas, divisiones entre elementos, etc).

Para la condición del *if()* hay varios vectores definidos: uno de ellos (*cond1*) con valores muy fácilmente predecibles, *cond3* usa valores aleatorios, dificilmente predecibles, y *cond2* usa valores predecibles (forman un patrón), así que puede preguntar por el valor de los elementos de esos vectores. Ver la fase de inicialización de vectores en el código fuente para más detalles (y para cambiar el patrón en *cond2*).

Pruebe a desactivar el uso de instrucciones mejorado (/arch:IA32 en lugar de /arch:SSE). ¿A qué se deben las diferencias en los tiempos observados, sobre todo cuando se usa una condición dificilmente predecible por la BTB?

4) Sumatorio de los elementos de un vector. Probar al menos con:
   a. Una sola suma por iteración (sumatorio).
   b. Dos sumas por iteración. Por ejemplo, la suma de dos vectores sobre el mismo acumulador, la suma de una constante y de un vector. Atención: se puede escribir de varias formas, por ejemplo: *a)* z=z+(a[i]+b[i]); *b)* z=(z+a[i])+b[i]; etc. O bien: *a)* z=z+(a[i]+3.0); *b)* z=(z+a[i])+3.0; etc.
   c. Una suma condicional por iteración. Es decir, en cada iteración, al acumulador se le sumará el elemento de un vector o el de otro en función de que una condición sea cierta o falsa. Probar con dos tipos de condiciones del *if()*:
      i. La condición sea difícilmente predecible (p. ej. Random). Cuidado: no introducir la llamada a *función rand()* (o similar) en el cuerpo del bucle (tarda mucho).
      ii. La condición sea fácilmente predecible por la BTB.

   Para la condición del *if()* hay varios vectores definidos: uno de ellos (*cond1*) con valores muy fácilmente predecibles, *cond3* usa valores aleatorios, dificilmente predecibles, y *cond2* usa valores predecibles (forman un patrón), así que puede preguntar por el valor de los elementos de esos vectores. Ver la fase de inicialización de vectores en el código fuente para más detalles (y para cambiar el patrón en *cond2*).

   Pruebe a desactivar el uso de instrucciones mejorado (/arch:IA32 en lugar de /arch:SSE). ¿A qué se deben las diferencias en los tiempos observados, sobre todo cuando se usa una condición dificilmente predecible por la BTB?

Notas (para todos los problemas)

- Evitar que se produzcan desbordamientos (underflow o overflow) en las variables *float*. Eso genera una excepción aritmética que añade muchísimo tiempo de ejecución.
- Ejecutar primero en modo debug para detectar errores. Sin embargo, los resultados finales de tiempo deben ser ejecutados en modo **Release**. Cuidado: pensar si debe usarse /Ox o /Od.
- Considere ejecutar el código en otro compilador o sistema operativo, pero a ser posible, en el mismo ordenador. Acuda a tutorías si tiene problemas para portar parte del código. Recuerde usar flags del compilador que ofrezcan un comportamiento equivalente a: *"Full Optimization (/Ox), Favor Fast Code (/Ot), Fast Floating Point"*
- Recuerde las opciones de compilación de la práctica anterior (ver boletín).
- Para ver el código fuente durante la ejecución, recuerde insertar un punto de ruptura y ejecutar el programa con F5. Sin embargo, para medir tiempos, el programa debe ejecutarse sin depurar, con Ctrl-F5, y con los siguientes flags del compilador:
  - Asegúrese de que el modelo de punto flotante "Fast" '*/fp:fast*' está seleccionado: (*"Propiedades del proyecto→ Propiedades de configuracion → C/C++ → Generación de Código → Modelo de punto flotante → Rápido /fp:fast*).
  - Usar instrucciones SSE *'/arch:SSE'*: (*"Propiedades del proyecto→ Propiedades de configuracion → C/C++ → Generación de Código → Habilitar conjunto de instrucciones mejorado → Extensiones SIMD de streaming (/arch:SSE)*).
  - Asegúrese de activar este flag: *"Propiedades del proyecto--> Propiedades de configuracion --> C/C++ --> Optimizacion --> Expansion de las funciones inline --> Solo __inline (/Ob1)"*. Esto evita que el código de las funciones se inserte dentro de *main()*.
- El proyecto que se da no usa OpenCV y por tanto, debería funcionar en cualquier PC Intel (no funciona correctamente en procesadores AMD). El código hace uso de la instrucción *rdtsc* para medir ciclos de reloj. Tenga en cuenta que en AMD esta instrucción puede no existir, o tener un comportamiento diferente.
- La **duración del código real dependerá de la máquina concreta** y del estado de carga de la memoria y CPU. Se aconseja no tener abiertas muchas aplicaciones y ejecutar varias veces para quedarse con el mínimo tiempo.
- Asegúrese de que las opciones de energía no estén configuradas para el modo de "Ahorro de energía", ya que en este modo la velocidad de la CPU es variable. Puede cambiarse el modo de energía desde la linea de comandos con la orden *powercfg.cpl* (Pulse *Windows - R*, y teclee *powercfg.cpl*).
- Toda función debe incluir una sentencia return (por ejemplo, *return a[N_ITER-1]; o return z; para los bucles que tengan algún tipo de acumulador*). El valor devuelto debe imprimirse en pantalla desde la función *main()*. De esa manera evitaremos que el compilador suprima la función.

PRESENTACIONES DE LOS RESULTADOS DE LA PRÁCTICA.

Se habilitará una Actividad en EV para que uno de los componentes del grupo pueda subir el fichero de presentación de su trabajo, de forma que antes de comenzar la sesión de presentaciones, todos los trabajos deberán estar disponibles para que el profesor los disponga en el PC conectado al proyector. *Para nombrar el fichero, use el nombre y apellidos de uno de los componentes del grupo*.

En la sesión de presentación, se sorteará qué alumno de cada grupo presenta primero y cuál después (cada uno la mitad). Si el grupo fuera de 3 alumnos, el tercero se quedará esperando las preguntas de los profesores, o para repetir alguna de las transparencias que hay quedado menos clara. Por tanto, todos los alumnos deben prepararse la presentación entera.

Se proporciona a los alumnos en ev.us.es una plantilla sobre cómo se van a evaluar las presentaciones (matriz de evaluación).