ASD: ARQUITECTURA DE SISTEMAS DISTRIBUIDOS. 3º GTI.
PRÁCTICA 3B: PROCESADORES SUPERESCALARES: LIMITES DEL ILP

OBJETIVOS.

En esta práctica los alumnos estudiarán los límites de ILP que se pueden conseguir con técnicas dinámicas en procesadores de propósito general (GPP). Para ello medirán el tiempo de ejecución de un código en C++ en una máquina GPP real. Se estudiarán diferentes variaciones, en alguna de las cuales prácticamente se alcanza el límite del flujo de datos (*data-flow limit*), de manera que el tiempo medio por iteración de un bucle es debido a tal límite. Los alumnos deberán analizar cada código para razonar cuál es su ruta crítica y entender el motivo de las diferencias de rendimiento entre las diferentes variaciones.

PREPARACIÓN.

- Se da una función *test1()* en código en C que resuelve un producto múltiple con una variable acumuladora. Es un bucle simple para ver el máximo IPC que se puede conseguir. El código se ha escrito lo más simple posible para que pueda probarse en cualquier compilador:

```
double test1 (int niter)  {
    int i;
    double acc=1.0;
    for (i=0; i<niter; i++)
    {
            acc=acc*a[i];
    }
    return acc;
}
```

- Los cálculos se repetirán con vectores de distintos tamaños, para estudiar el impacto del acceso a memoria en el rendimiento, según el tamaño de los mismos. Es decir, `niter` cambiará creciendo en potencias de 2 así: 512, 1024, 2048,...
- ¿Cual será el diagrama de del flujo de datos y la operación(es) en el camino crítico para el estacionario del bucle del *test1()*?
- Cada ejecución se repite 64 veces, de manera que si el vector (o los vectores) caben en un caché, en la primera repetición los vectores quedarían alojados en tal caché para todas las demás repeticiones. Sin embargo, si el vector (o los vectores) no cupiesen en un caché, entonces la CPU debería ir a buscarlos en todas las repeticiones al nivel inferior, incurriendo en cierta tasa de errores de acceso (Miss Rate).
- El alumno debe calcular el rango de número de elementos en los que el vector cabe en los diversos niveles de caché. El tamaño de cachés de datos para las aulas A4.30 y A4.31 es: L1: 32 KB , L2: 256 KB , L3: 6 MB
- El alumno debe hallar la tasa de errores de acceso (Miss Rate) siendo el tamaño de línea es de 64B. Notar que el acceso a los elementos del vector es siempre consecutivo (puede además suponer que la dirección de a[0] es múltiplo de 64).
- El Ancho Banda de RAM pedido por la CPU puede ser grande. Por ejemplo, para una CPU a 3 GHz, que ejecuta una iteración del *test1*() en 2 ciclos, ¿cuanto vale el Ancho Banda de RAM pedido por la CPU en GB/s?

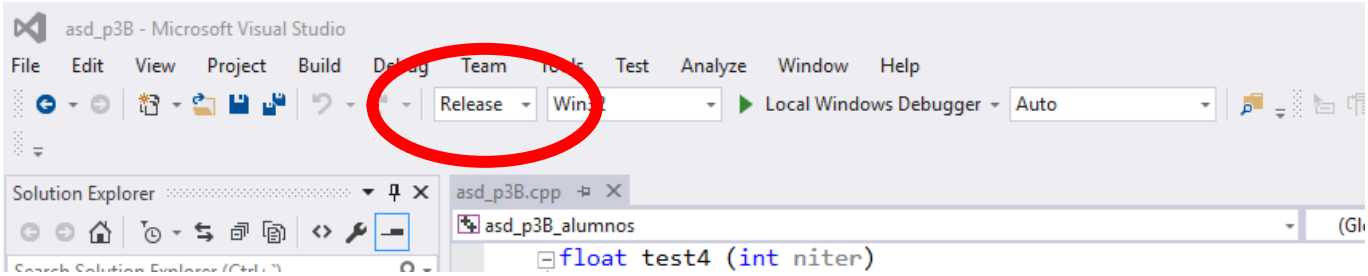
REALIZACIÓN DE LA PRÁCTICA.

- Al ejecutar en un GPP los programas reales (que se pueden descargar de ev.us.es), evidentemente se van a obtener diferentes resultados de ciclos según la máquina donde se pruebe. *(NOTA: los programas están preparados para Visual Studio; si se usara el código fuente con otro compilador, no olvidar activar opciones similares a: "Optimización completa (/Ox) , Favorecer código rápido (/Ot), Modelo de Punto Flotante: Fast" .*
- Para facilitar la ejecución, se proporciona un proyecto (o "solución") creado con Visual Studio 2015.
- La instrucción de medida de ciclos rdtsc funciona en todos los procesadores Intel (pero no en algunos de AMD).
- Si se quisiera medir tiempos en segundos, habría que modificar el valor de la constante siguiente en el fichero de encabezado (cabecera, *header*): *QueryPerformanceTiming_rdtsc.h*:

  #define CPU_FREQ_HZ (3.20e+9)

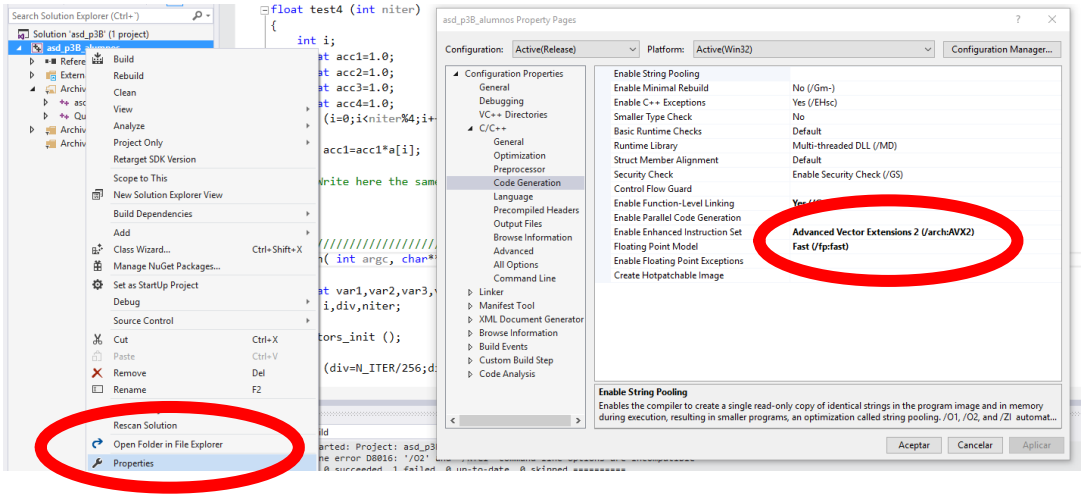
  //write here your CPU frequency

  Si no se hiciera el tiempo en segundos se calcularía mal.
- Para hacer esta prueba, no use combinaciones de energía del estilo de "Administración de energía mínima", pues puede ocasionar problemas en la medición de tiempos (la frecuencia de la CPU es variable en este modo). En Windows, haga clic en Inicio y en Ejecutar, escriba powercfg.cpl y haga clic en Aceptar. Seleccione una combinación de energía distinta de Administración de energía mínima en el cuadro Combinaciones de energía.
- Hay que ejecutar sin depuración (tecla Crtl F5) para que la ventana de comando no se cierre, o bien ejecutar en una consola.
- Debe usarse el modo Release (que está preparado con todas las optimizaciones). Ver figura:



Asegurarse de elegir en las propiedades del proyecto:

- el modelo Fast de coma flotante
- y las instrucciones del núcleo SIMD AVX2 (se estudiarán en teoría más adelante), como se muestra en la figura.



Ejecute todo el proyecto VS2015 (ver ev.us.es) en modo Release. El proyecto que se da no usa OpenCV y por tanto, debería funcionar en cualquier PC Intel (no funciona correctamente en procesadores AMD).

Evidentemente 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.

Se deben corroborar los **datos previos sobre tamaño de caché y tipo de CPU**:

- Máquina (sobre todo procesador) donde se ejecuta el código real. Se puede consultar en el panel de control --> Sistema. También se puede usar el programa CPU-Z (se puede 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.

Resultados: Habilitar las optimizaciones según se indicó anteriormente y:

- Ejecutar el proyecto en modo release y anotar los CPI, y los ciclos por iteración de test1().
- Dibujar un esquema de tal ruta crítica.
- Tratar de razonar cuál es la operación u operaciones que están en la ruta crítica, o la circunstancia que hace que el CPI sea mayor que 1/4, o que la duración de una iteración sea superior a *Numero_instrucciones_por_iteracion_de_ensamblador* / *m* (*m*=grado superescalaridad = 4 en la mayoría de las CPU superescalares de alto rendimiento).
- Dibujar un cronograma temporal simplificado de cómo se están ejecutando las operaciones. Considerar que hay 2 UF de MULT (que permiten segmentación).
- Indicar la latencia de la unidad funcional correspondiente a esa operación u operaciones, o en su caso, la circunstancia que conduce a una duración mayor.
- Analizar qué ocurre cuando crece el número de iteraciones. Tratar de razonar el motivo por el que la pérdida de rendimiento no se produce de igual manera en las distintas funciones.
- Para saber si las instrucciones del núcleo SIMD AVX2 (se estudiarán en teoría más adelante), ponga un punto de ruptura antes del bucle, mire el ensamblador, busque la instrucción de salto hacia atrás (seguramente `jl`) y compruebe que instrucciones contiene para hacer los productos. Puede ocurrir que el VS haya vectorizado, entonces cada instrucción ejecuta varias operaciones en paralelo (es como un desenrollado pero en una sola instrucción). Por ejemplo:
  1. Si empiezan por v y acaban por ps o pd, el VS ha vectorizado. Serían instrucciones: vaddps, o vmulps, que trabajan con 8 float totalmente en paralelo (registros de 256 bits). O instrucciones vaddps , vmulpd, que trabajan con 4 double totalmente en paralelo (registros de 256 bits).

2. Si empiezan por v pero acaban por ss o sd, el VS no ha vectorizado. Serían instrucciones: vaddss, o vmulss, que trabajan con 1 float . O instrucciones vaddss , vmulsd, que trabajan con 1 double.

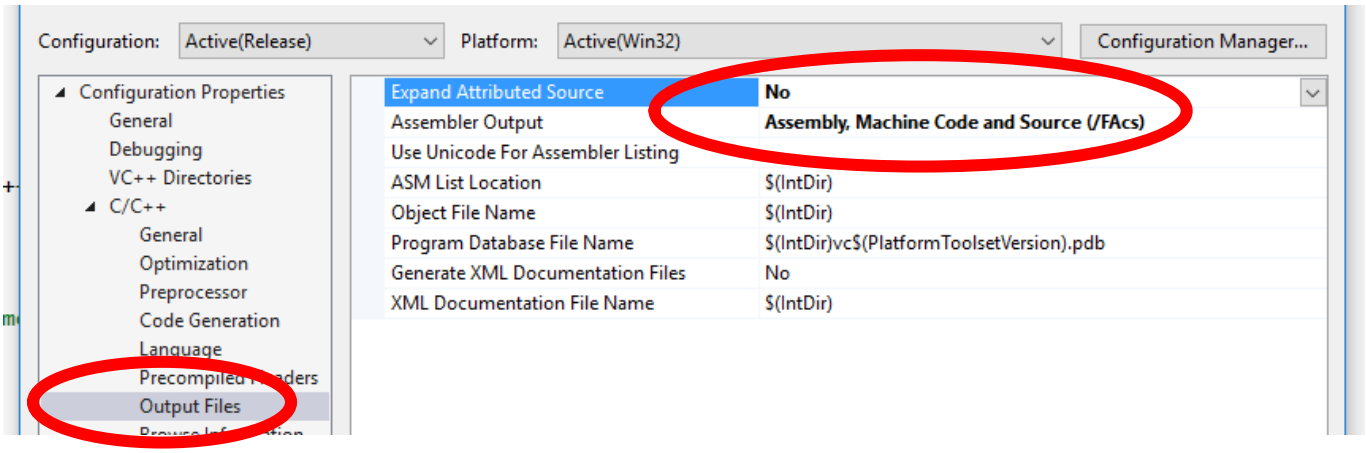
A continuación escribir el producto múltiple con 2,3,4, variables acumuladoras **rellenando el código que falta en test2(), test3(), test4().** Repetir los pasos anteriores.

IMPORTANTE: AL FINAL, NO DEBE SALIR NINGÚN MENSAJE PARA NINGUNO DE LOS TESTS QUE INDIQUE ERROR EN LOS RESULTADOS (ALGO COMO: `"!!!  ERROR IN test3() CODE . TIMING RESULTS NOT VALID FOR test3()."`)

Finalmente **sustituya el producto por la suma** en todos los test y comprueba que ensamblador genera VS. Si reemplaza en el código el texto *a[por el texto +a[, (y también con espacio: * a[por el texto + a[ ) se cambiarán todas las operaciones. Indique lo más relevante de los nuevos valores de ciclos por elemento del vector.

APÉNDICE: DETALLES SOBRE EL CÓDIGO A TENER EN CUENTA PARA HACER MÁS PRUEBAS (POR EJEMPLO EN PRACTICA SIGUIENTE):

- Para poder ver el código ASM o depurar más cómodamente compruebe que esta opción está activa: *"Propiedades del proyecto--> Propiedades de configuracion --> C/C++ --> Optimizacion --> Expansion de las funciones inline --> Solo __inline (/Ob1)"* (existen opciones similares en otros compiladores). De esa forma, las funciones test1(), test2() etc no se "empotran " en el código del main(), sino que tienen su cuerpo independiente y se llaman (CALL) desde el main(). Si tal opción no está activa, el compilador podría expandir todo el código de una función dentro del ASM del maín () y luego reordenar como le parezca, con lo cual el ASM no es fácil de seguir ni entender.
- Para no tener errores de compilación y para poder ver el fichero ASM (será algo como "asd_p3B.cod" en la carpeta Release), usar:



- Cada función tiene una sentencia return (p ej : return a[MAX_NUM_ELEM-1];). Dejar siempre esta sentencia y las impresiones de las variables devueltas. Si no se usa la variable de salida, el compilador podría eliminar totalmente la llamada a la función.

APÉNDICE: CÓDIGO FUENTE PARA LAS PRUEBAS

```
//-----------------------------------------------------
////////////////////////////////////////////////////////////////////////////////////////////////////
//  Dpto ATC. www.atc.us.es
//  ASD subject 3º GTI
//  version 1. Nov. 2012. Fernando Diaz del Rio
//  version 2. Mar. 2014. Jesus Rodriguez Leal
//  - Doing a  test to reach ILP limits in superscalar processor
//  - TEST TIMING OF simple routine
//-----------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#include <iostream>
using namespace std;
//-------------------------------------------------------------------
// project def and includes
// new version of QueryPerformanceTiming , to measure CPU cycles more precisely using rdtsc asm instr.
#include "QueryPerformanceTiming_rdtsc.h"

#define  N_REPETIC 64

// repeat several times each test to extract the minimum time
// and to have the caches filled
#define  MAX_NUM_ELEM (4*1024*1024)

double a[MAX_NUM_ELEM ];

////////////////////////////////////////////////////////////////////////////////////////////////////
// vectors  init
void vectors_init ()
{
        int i;
        for (i=0;i<MAX_NUM_ELEM;i++)
        {
                // a[i] must have near to 1 values to avoid overflow/underflow in tests
                a[i]=1.0f;
        }
}
////////////////////////////////////////////////////////////////////////////////////////////////////
/// tests
double test1 (int niter)
{
        int i;
        double acc=1.0;
        for (i=0;i<niter;i++)
        {
                acc=acc*a[i];
        }
        return acc;
}

double test2 (int niter)
{
        int i;
        double acc1=1.0;
        double acc2=1.0; // initialization of accumulator variables
        for (i=0;i<niter%2;i++)  //this is a prolog code to check if niter is not multiple of 2
        {
                acc1=acc1*a[i];
        }
        // Write here the same algorithm using two accumulators
}
double test3 (int niter)
{
        int i;
        double acc1=1.0;
        double acc2=1.0;
        double acc3=1.0;
        for (i=0;i<niter%3;i++)    //this is a prolog code to check if niter is not multiple of 3
        {
                acc1=acc1*a[i];
        }
        // Write here the same algorithm using 3 accumulators
}
```

```
double test4 (int niter)
{
        int i;
        double acc1=1.0;
        double acc2=1.0;
        double acc3=1.0;
        double acc4=1.0;
        for (i=0;i<niter%4;i++)    //this is a prolog code to check if niter is not multiple of 4
        {
                acc1=acc1*a[i];
        }
        // Write here the same algorithm using 4  accumulators
}
////////////////////////////////////////////////////////////////////////////////////////////////////
int main( int argc, char** argv ) {

double var1,var2,var3,var4;
int i,div,niter;
vectors_init ();
for (div=MAX_NUM_ELEM/512;div>=1;div/=2)
        {
        niter = MAX_NUM_ELEM / div;

        QPTimer c1,c2,c3,c4;
        c1.Calibrate(); //calibrates timer overhead and set cronometer to zero
        c2.Calibrate(); //calibrates timer overhead and set cronometer to zero
        c3.Calibrate(); //calibrates timer overhead and set cronometer to zero
        c4.Calibrate(); //calibrates timer overhead and set cronometer to zero

        // measuring tests
        for (i=0;i<N_REPETIC;i++)
        {
                c1.Start();  // start timer
                var1=test1(niter); // Do the test
                c1.Stop();  // stop timer
                c1.Reset();

                c2.Start();  // start timer
                var2=test2(niter); // Do the test
                c2.Stop();  // stop timer
                c2.Reset();

                c3.Start();  // start timer
                var3=test3(niter); // Do the test
                c3.Stop();  // stop timer
                c3.Reset();

                c4.Start();  // start timer
                var4=test4(niter); // Do the test
                c4.Stop();  // stop timer
                c4.Reset();
                }
                // end of    testing
        cout << endl << "ONLY PRINTING OUTPUT VARIABLE TO PREVENT THAT THE COMPILER ELIMINATES FUNCTION CALLS: "
                        << var1 << ", " << var2 << ", " << var3 << ", " << var4 << endl;
        cout << endl << "-Number of iterations MAX_NUM_ELEM: " << niter << endl;
        cout << endl << "-Number of measures: " << c1.NumberOfMeasures() << endl;
        c1.PrintMinimumCyclesByIteration(" Minimum time in cycles for an iteration of test1() is:", niter);
        c2.PrintMinimumCyclesByIteration(" Minimum time in cycles for an iteration of test2() is:", niter);
        c3.PrintMinimumCyclesByIteration(" Minimum time in cycles for an iteration of test3() is:", niter);
        c4.PrintMinimumCyclesByIteration(" Minimum time in cycles for an iteration of test4() is:", niter);
                cout << endl  ;
        }
}
```

página dejada intencionadamente en blanco

TABLA DE RESULTADOS
ASD: ARQUITECTURA DE SISTEMAS DISTRIBUIDOS. 3º GTI.
PRÁCTICA 3B: SUPERESCALARES: LIMITES DEL ILP

ALUMNO:	
FECHA Y HORA:	

PREPARACIÓN:

Para test1(): Dibujar un esquema del flujo de datos. Señalar Operación(es) en la ruta crítica.

Ancho Banda de RAM pedido por CPU:

Calcular rango de número de elementos en los que el vector cabe en : L1: 32 KB , L2: 256 KB , L1: 6 MB
L1: L2: L3:

Tasa de errores de acceso (Miss Rate) si el tamaño de línea es de 64B.

RESULTADOS:

Procesador:	**Frecuencia de reloj:**	**Tamaño de cachés L1:**	**L2:** **L3:**

	Tiempo mínimo: ciclos por elemento. Anotar 4 rangos (L1,L2,L3, RAM)	**Ensamblador del bucle**	CPI	**Dibujar un cronograma simplificado. Pensar como se halla la duración de la UF en la ruta crítica (en ciclos) al desenrollar.**
test1( )	L1: L2: L3: RAM:			
test2( )	L1: L2: L3: RAM:			

test3( )	L1: L2: L3: RAM:			
test4( )	L1: L2: L3: RAM:			

Bucles largos con vectores grandes. Explicar el comportamiento del código cuando el tamaño del vector es elevado (anotar a partir de que MAX_NUM_ELEM hay cambios en los tiempos). :

test1()

test2()

test3()

test4()

Sustituir el producto por la suma . Indique lo más relevante de los nuevos valores de ciclos por elemento del vector.