% Evaluación de la práctica 1: Fundamentos de Vectorización en x86
62222 Computación de Altas Prestaciones
Máster Universitario en Ingeniería Informática % Jesús Alastruey Benedé
Área Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza % 31-octubre-2020

Resumen

Para la evaluación de la práctica 1 vais a resolver algunas cuestiones correspondientes a los puntos 1.4, 1.6 y 2.2 del guión de prácticas. Los tiempos y métricas deberán obtenerse para las máquinas de los laboratorios L0.04 o L1.02. Sed concisos en las respuestas. Se valorarán las referencias utilizadas.

Parte 1. Vectorización automática

4. ¿Cuántas instrucciones se ejecutan en el bucle interno (esc.avx, vec.avx, vec.avxfma y vec.avx512)?

```
    for (int i = 0; i < LEN; i++)
        x[i] = alpha*x[i] + beta
```

Calcula la reducción en el número de instrucciones respecto la versión esc.avx.

versión	icount	reducción(%)	reducción(factor)
esc.avx	6144	0	1.0
vec.avx	768	87.5	0.875
ec.avxfma	768	87.5	0.875
ec.avx512	384	93.7	0.937

Para el calculo del ICOUNT se uso la formula ICOUNT = N × LEN donde N es el numero de intrucciones en el cuerpo del bucle(6) y LEN el numero de iteraciones del bucle(1024 inicialmente) este ultmo valor es muy importante dado que el numero de iteraciones va ha ser determinada por el numero de bits que pueda ser procesado por el procesador. Calculemos un LEN como ejemplo para el vec.avx512: Sabemos que este procesador trabaja con 512b y estamos usando un tipo float de 32b para las operaciones. Por lo que al dividir 512/32 obtenemos que podemos operar con 16 elementos a la vez. Si dividimos el numero de iteraciones entre 1024/16 obtendremos las iteraciones que se ejecutaran con este procesador que puede vectorizar usando 512b. Obtenemo de esta forma que el valor de LEN para este caso es 64. Ahora si remplazamos los valores en la ecuacion mencionada previamente obtenemos:

```
vec.avx512: ICOUNT = N × LEN = 6 × 64 = 384
```

Para el calculo de la reduccion tomando el mismo ejemplo de vec.avx512:

```
vec.avx512:reduccion = valor_original - nuevo_valor = 6144 - 384 = 5760

vec.avx512: reducción(factor) = reduccion / valor_original = 5760/6144 =
0.9375

vec.avx512: reducción(%) = reducción(factor) * 100 = 93,7%
```

5. A partir de los tiempos de ejecución obtenidos [...], calcula las siguientes métricas para todas las versiones ejecutadas:

   - Aceleraciones (*speedups*) de las versiones vectoriales sobre sus escalares (vec.avx y vec.avxfma respecto esc.avx). Para el calculo de speedup usaremos la siguiente formula con un calculo de ejemplo:

     ```
     vec.avx: Speedup = tiempo_scalar/tiempo_vectorial = 465.9/70.5 = 6.60
     ```

   - Para los calculos a continuacion necesitaremos calcular IPS(Intrucciones por segundo) Y FLOPS(intrucciones punto flotante por segundo).

     ```
     IPS = instruccion/segundo = 6144 * 10^6 /465.9 = 13187379.266 inst/s

     Validamos en el codigo cuantas intrucciones de punto flotante se
     ejecutan:
     4 operaciones punto flotante(3 sumas y una multiplicacion)
     el calculo total es 4*1024

     Primero validamos en el codigo del bucle interno cuantas intrucciones
     de punto flotante tenemos, vemos que son 4 intrucciones de punto
     flotante por 1024 iteraciones en el caso del escalar lo que da un total
     de  4096, como el tiempo que se obtiene en la ejecucion es en
     nanosegundos se aplica una regla de 3 para determinar el numero de
     intrucciones en segundos.

     FLOPS = instruccion_flotantes/segundo = 4096 * 10^6/465.9 = 8791586.177
     ```

   - Rendimiento (R) en GFLOPS. Una vez que calculamos los FLOPS la formula a usar para el calculo de GLOPS seria:

     ```
     GFLOPS = FLOPS/10^9 = 8791586.177 / 10^9 = 0.00879 Billones de
     intrucciones flotantes por segundo
     ```

   - Rendimiento pico (R~pico~) teórico de un núcleo (*core*), en GFLOPS. Para las versiones escalares, considerar que las unidades funcionales trabajan en modo escalar. Considerar asimismo la

capacidad FMA de las unidades funcionales solamente para las versiones compiladas con soporte FMA.

- Velocidad de ejecución de instrucciones (V~I~), en Ginstrucciones por segundo (GIPS).

  Una vez que calculamos los IPS la formula a usar para el calculo de GIPS seria:

  ```
  GIPS = IPS/10^9 = 13187379.266 / 10^9 = 0.0131 Billones de intrucciones por segundo
  ```

esc.avx ips 13187379.265936896 esc.avx gips 0.013187379265936897 esc.avx flop 8791586.177291265 esc.avx gflop 0.008791586177291266

---

vec.avx ips 10893617.021276595 vec.avx gips 0.010893617021276595 vec.avx flop 7262411.347517731 vec.avx gflop 0.007262411347517731

---

vec.avxfma ips 9588014.981273409 vec.avxfma gips 0.009588014981273409 vec.avxfma flop 6392009.987515606 vec.avxfma gflop 0.006392009987515606

---

versión	tiempo(ns)	speed-up	R(GFLOPS)	~pico~(GFLOPS)	V~I~(GIPS)
esc.avx	465.9	1.0			
vec.avx	70.5	6.60			
vec.avxfma	80.1	5.81			

¿La velocidad de ejecución de instrucciones es un buen indicador de rendimiento?

No me queda claro si con velocidad quiere decir a que tan rapido es en tiempo???

Actualmente existen varias metricas de rendimiento la usada con mayor frecuencia es el tiempo de ejecución. Usamos este para evaluar si una arquitectura es optima o tiene mejor rendimiento respecto a a otra.Se debe tomar en cuenta que esta no es la unica medidas tamien se podrian usar MIPS Y GFLOPS. El problema de usar

MIPS es que dependiendo de la arquitectura del procesador(RISC, CISC) y su ISA(Instruction Set Arquitecture) el numero de intrucciones de un fragmento de codigo estara condicionado por esto. Sin embargo con los GFLOPS o numero de operaciones en punto flotante va hacer la misma sin importar la arquitectura por lo que diria que es mas confiable.

Parte 2. Vectorización manual mediante intrínsecos

2. Escribe una nueva versión del bucle, `ss_intr_AVX()`, vectorizando de forma manual con intrínsecos AVX. Lista el código correspondiente a la función `ss_intr_AVX()`.

   Analiza el fichero que contiene el ensamblador de dicha función y busca las instrucciones correspondientes al bucle en `ss_intr_AVX()`.
   ¿Hay alguna diferencia con las instrucciones correspondientes al bucle en `scale_shift()` (versión vec.avx)?
   ¿Hay diferencia en el rendimiento de las funciones `scale_shift()` (versión vec.avx) y `ss_intr_AVX()`?