Práctica 1: Optimización de una aplicación secuencial

Jose Antonio Lorencio Abril

1. Detalla las características relevantes del ordenador elegido (microarquitectura de CPU y memoria).

Primero vamos a ver las características de nuestra máquina. Usamos el comando *lscpu* y la orden *lshw*, de la que nos interesa la salida correspondiente a la CPU. Vamos a *wikichip.com* y buscamos nuestra CPU: i5-7200U. Aquí encontramos la microarquitectura: **Kaby Lake**. Entonces es un Intel® Core^{TM} i5 CPU 2.5 GHz (hasta 3.1 GHz en modo turbo), con 2 núcleos, con HyperThreading de 2 hilos por núcleo. Tiene 128KiB de caché L1 (separada en dos cachés de 64KiB), 512KiB de caché L2 y 3MiB de caché L3 compartida. Tiene ISA soportado x86-64. Vemos, además, que tiene AVX y AVX2.

Para conocer el SO, usamos la orden *hostnamectl*, y vemos que tenemos Ubuntu 20.04.1 LTS.

2. ¿Cuál es el rendimiento pico que puede alcanzar teóricamente usando coma flotante de simple precisión (en GFLOPS)?

Para saber las unidades funcionales en punto flotante que permiten trabajar vectorización, entramos en la microarquitectura Kaby Lake en wikichip, y nos fijamos en el diseño de esta.

Vemos que hay dos unidades de esta clase (las que tienen FP FMA).

Por tanto, tenemos que:

- $\frac{flops}{instruction} = 2$, porque soporta FMA
- $\frac{operations}{instruction} = 8$, porque soporta AVX
- $\frac{instructions}{cycle} = 2$, por tener 2 unidades con soporte FMA
- $frec = 3.1GHz$, la turbo porque buscamos el pico
- $\frac{cores}{socket} = 2$ (indicado en la salida de *lscpu*)
- $\frac{sockets}{node} = 1$

De modo que el rendimiento pico es:

$$RendimientoPico = 2 \cdot 8 \cdot 2 \cdot 3.1 \cdot 2 = 198.4 \frac{Gflops}{s}$$

3. ¿Cuál es el ancho de memoria pico que puede alcanzar teóricamente?

Pasamos ahora al cálculo del ancho de banda pico, usamos $lshw\ -class\ memory$.

- $ControladorMemoria = 1$
- $\frac{Canales}{Control} = 1$, aunque parezca que tiene dos (los bank), el segundo no se utiliza
- $Bytes = 8$, la memoria es de 8GiB
- $Freq = 2.133GHz$

Por lo que el ancho de banda pico es:

$$BandWidthPico = 1 \cdot 1 \cdot 8 \cdot 2.133 = 17.1GB/s$$

4. Mide el rendimiento pico en coma flotante de simple precisión del ordenador elegido utilizando los benchmarks helloflops1 y helloflops2. Explica los resultados obtenidos.

Pasamos ahora a ver cuán cerca del rendimiento pico podemos ir, con unos benchmarks de prueba.

Ejecutando el primero, $helloflops1$, obtenemos:

```
Initializing
Starting Compute
GFlops =     128.000, Secs =       1.695, GFlops per sec =      75.496
```

Obtenemos siempre alrededor de 75 GFlops, lo que representa un ~75% del rendimiento pico para un único núcleo $\left(\frac{198\ rendimiento\ pico\ total}{2\ nucleos} \simeq 99\ GFlops/s\right)$. Este resultado es bastante bueno, pues parece que la potencia por núcleo se aprovecha bastante.

Ejecutando $helloflops2$:

```
Starting Compute on 4 threads
GFlops =     512.000, Secs =       4.634, GFlops per sec =     110.499
```

Obtenemos siempre un valor alrededor de 112 GFlops, teniendo, por tanto, un 56% del rendimiento pico. Vemos aumentamos los GFlops/s, pero se reduce, de alguna forma, el aprovechamiento de cada núcleo al paralelizar. Esto debe ser porque la comunicación entre hilos no es la idónea. Por tanto, a pesar de las buenas noticias respecto al rendimiento por núcleo, no podemos (al menos únicamente con esta información) estar tan contentos con la paralelización.

5. Mide el ancho de banda pico de memoria del ordenador elegido utilizando el benchmark copy. Explica el resultado obtenido.

No lo hemos hecho con *copy*, sino con *hellomem*, que es el que tenemos cómo compilar.

```
Initializing
Starting BW Test on 4 threads
Gbytes =    1024.000, Secs =      89.802, GBytes per sec =      11.403
```

Estos 11.4 GB/s (siempre sale alrededor de este valor) representan un 67% del ancho de banda pico, lo cual parece un rendimiento aceptable, sobre todo comparado con el resultado proporcionado por los profesores, que es similar.

6. Usando los benchmarks sumaflopsX, calcula la latencia de las sumas en simple precisión del ordenador elegido y trata de determinar de cuántas unidades funcionales dispone cada core para la suma en simple precisión.

Como pone en la práctica, debemos tener en cuenta que prescindimos de la vectorización, por lo que el nuevo rendimiento es de 3.1 GFlops/s.

En $sumaflops1$ obtenemos 0.272 GFlops/s, que es un ~10% del rendimiento máximo, por tanto, la latencia podemos suponerla de unos 10 ciclos de reloj.

Vemos como, al igual que en el boletín, aumenta de forma lineal al aumentar las variables. Y, también, al igual que en el boletín, crece linealmente hasta tener 8 variables, a partir de aquí se estabiliza. Por tanto, como tenemos 2 UF por núcleo, en realidad la latencia es 4, es decir, que al usar pocas variables desaprovechamos muchísimo nuestra CPU.

7. Valoración de la práctica

La práctica me ha parecido interesante, sobre todo porque hemos visto muchos de los conceptos que se tratan en teoría, en nuestra propia máquina. Esto me parece muy positivo, porque nos hace ponernos en circunstancia y situar la teoría dentro del mundo real. Como aspecto negativo solo puedo decir que el compilador gcc a veces hace cosas extrañas, y esto me ha provocado mucha confusión, pues obtenía resultados que no comprendía. Poco podemos hacer contra esto, pero quizás hubiera estado bien un aviso por parte del profesorado.