Evaluación de la práctica 2: Limitaciones a la Vectorización
30237 Multiprocesadores - Grado Ingeniería Informática
Esp. en Ingeniería de Computadores Universidad de Zaragoza

Sergio García Esteban

14-marzo-2020

Resumen

Los tiempos y métricas deberán obtenerse para máquinas de los laboratorios L0.04 o L1.02. Sed concisos en las respuestas. Se valorarán las referencias utilizadas.

Notas generales

El trabajo puede presentarse de forma individual o en grupos de máximo dos personas. Podéis trabajar en grupos mayores, pero **cada grupo debe elaborar el material a entregar de forma independiente**. Hacedme llegar vuestros trabajos **en formato pdf** a través de la entrega habilitada en la web de la asignatura (moodle). Incluid vuestro nombre y apellidos en la cabecera del documento y vuestro NIP en el nombre del fichero (p2_NIP.pdf).
Plazo límite de entrega: jueves, 19 de marzo, 23h59m59s.

Parte 1. Efecto del alineamiento de los vectores en memoria

La función `ss_align_v1()` calcula el kernel *scale and shift*. El vector `x[]` está alineado con el tamaño vectorial de AVX, es decir, su dirección inicial es múltiplo de 32 bytes (256 bits).

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

La función `ss_align_v2()` hace el mismo cálculo pero con el vector `x[]` **NO** alineado, ya que se procesa desde el elemento con índice 1:

```
for (unsigned int i = 0; i < LEN; i++)
    x[i+1] = alpha*x[i+1] + beta;
```

Las funciones `ss_align_v1_intr()` y `ss_align_v2_intr()` implementan con intrínsecos los bucles de las funciones `ss_align_v1()` y `ss_align_v2()` respectivamente. En el primer caso los accesos a memoria son alineados y en el segundo son no alineados. Todas estas funciones las compilamos en la sesión de prácticas con las versiones 9.2 y 7.2 de `gcc`.

1. Para cada instrucción de escritura en memoria ejecutada, indica su tipo -escalar(E)/vectorial (V)- y la dirección del dato al que accede. En caso de instrucción vectorial, especifica solamente la dirección del primer elemento. Supón que el vector `x[]` tiene 32 elementos de tipo `float`, y que está almacenado a partir de la dirección `0x6020c0`. Añade las filas que sean necesarias en las tablas.

   Notación: para cada instrucción vectorial, indica el número de elementos en el vector, y si dichos elementos deben estar alineados (A) o no (U).

`align_v1()`	tipo inst.	dirección
`vmovaps %ymm0,-0x20(%rax)`	V8A	`0x6020c0`
`vmovaps %ymm0,-0x20(%rax)`	V8A	`0x6020e0`
`vmovaps %ymm0,-0x20(%rax)`	V8A	`0x602100`
`vmovaps %ymm0,-0x20(%rax)`	V8A	`0x602120`

`align_v2_gcc7()`	tipo inst.	dirección
`vmovups %xmm0,0x2017cf(%rip)`	V4U	`0x6020c4`
`vmovss %xmm0,0x2017cb(%rip)`	E1	`0x6020d4`
`vmovss %xmm0,0x2017bb(%rip)`	E1	`0x6020d8`
`vmovss %xmm0,0x2017ab(%rip)`	E1	`0x6020dc`
`vmovaps %ymm0,-0x20(%rax)`	V8A	`0x6020e0`
`vmovaps %ymm0,-0x20(%rax)`	V8A	`0x602100`
`vmovaps %ymm0,-0x20(%rax)`	V8A	`0x602120`
`vmovss %xmm0,0x20273e(%rip)`	E1	`0x602140`

align_v2_intr()	tipo inst.	dirección
`vmovups %ymm0,-0x20(%rax)`	V8U	`0x6020c4`
`vmovups %ymm0,-0x20(%rax)`	V8U	`0x6020e4`
`vmovups %ymm0,-0x20(%rax)`	V8U	`0x602104`
`vmovups %ymm0,-0x20(%rax)`	V8U	`0x602124`

Parte 2. Efecto del solapamiento de las variables en memoria

1. Escribe los tiempos de ejecución en ns de los bucles ejecutados en las siguientes llamadas a funciones. Describe muy brevemente en la tabla las tareas realizadas u obviadas **en tiempo de ejecución**. Notación:
   - [n]S: [no] comprobar solapamiento
   - [n]A: [no] comprobar alineamiento
   - E/V: ejecución escalar/vectorial En caso de que no se efectúe alguna tarea (nX), indicar la razón.

llamada a función	tiempo (ns)	tareas
ss_alias_v1(&y[1], y)	3954.1	S A -> E
ss_alias_v1(y, x)	78.8	S A -> V
ss_alias_v2(&y[1], &x[1])	92.3	nS(restrict) A -> V
ss_alias_v2(y, x)	78.3	nS(restrict) A -> V
ss_alias_v3(&y[1], &x[1])	92.3	nS(pragma ivdep) A -> V
ss_alias_v3(y, x)	78.3	nS(pragma ivdep) A -> V
ss_alias_v4(y, x)	78.3	nS(restrict) nA(builtin_assume_aligned) -> V

Parte 3. Efecto de los accesos no secuenciales (stride) a memoria

1. Lista el código ensamblador correspondiente al bucle interno de la función `ss_stride_vec()`.
   ¿Cuántas instrucciones vectoriales hay en el cuerpo del bucle?
   Ayuda: utiliza las etiquetas al final de cada línea para identificarlas.

```
vmovaps (%rax),%ymm5
vshufps $0x88,0x20(%rax),%ymm5,%ymm1
vperm2f128 $0x3,%ymm1,%ymm1,%ymm2
add    $0x40,%rax
vshufps $0x44,%ymm2,%ymm1,%ymm0
vshufps $0xee,%ymm2,%ymm1,%ymm2
vinsertf128 $0x1,%xmm2,%ymm0,%ymm0
vmulps %ymm4,%ymm0,%ymm0
vaddps %ymm3,%ymm0,%ymm0
vmovss %xmm0,-0x40(%rax)
vextractps $0x1,%xmm0,-0x38(%rax)
vextractps $0x2,%xmm0,-0x30(%rax)
vextractps $0x3,%xmm0,-0x28(%rax)
vextractf128 $0x1,%ymm0,%xmm0
vmovss %xmm0,-0x20(%rax)
vextractps $0x1,%xmm0,-0x18(%rax)
vextractps $0x2,%xmm0,-0x10(%rax)
vextractps $0x3,%xmm0,-0x8(%rax)
cmp    $0x6030c0,%rax
jne    4007d8 <ss_stride_vec+0x48>
```

20 instrucciones, de las cuales 15 son vectoriales.

```
**OPTATIVO**. Detalla las operaciones realizadas por las instrucciones
vectoriales del bucle interno en ‘ss_stride_vec()‘.

vmovaps (%rax),%ymm5 -> lee de memoria 8 elementos del vector x[]
vshufps $0x88,0x20(%rax),%ymm5,%ymm1 -> mezcla los 8 elementos leídos en la
        instr anterior y los siguientes 8 elementos en memoria, para obtener
        los 8 elementos de índice par.
vperm2f128 $0x3,%ymm1,%ymm1,%ymm2 -> permutación para ordenar los elementos
vshufps $0x44,%ymm2,%ymm1,%ymm0 -> mezcla 1 para ordenar los elementos
vshufps $0xee,%ymm2,%ymm1,%ymm2 -> mezcla 2 para ordenar los elementos
vinsertf128 $0x1,%xmm2,%ymm0,%ymm0 -> inserta 128 bits (xmm) de la mezcla
        2 en el registro resultante de la mezcla 1 para obtener los
        8 elementos ordenados
vmulps %ymm4,%ymm0,%ymm0 -> 8 elementos resultantes * alpha
```

```
vaddps %ymm3,%ymm0,%ymm0 -> 8 elementos resultantes + beta
vextractps $0x1,%xmm0,-0x38(%rax) -> extrae de xmm0 y escribe en memoria
                                     elemento índice 2
vextractps $0x2,%xmm0,-0x30(%rax) -> extrae de xmm0 y escribe en memoria
                                     elemento índice 4
vextractps $0x3,%xmm0,-0x28(%rax) -> extrae de xmm0 y escribe en memoria
                                     elemento índice 6
vextractf128 $0x1,%ymm0,%xmm0 -> extrae de ymm0 a xmm0 los siguientes 4
                                 elementos
vextractps $0x1,%xmm0,-0x18(%rax) -> extrae de xmm0 y escribe en memoria
                                     elemento índice 10
vextractps $0x2,%xmm0,-0x10(%rax) -> extrae de xmm0 y escribe en memoria
                                     elemento índice 12
vextractps $0x3,%xmm0,-0x8(%rax)  -> extrae de xmm0 y escribe en memoria
                                     elemento índice 14
```

2. Calcula la aceleración (*speedup*) de la versión `icc` sobre la `gcc`.

   versión escalar: 470.1/470.2 -> 100% versión vectorial: 608.0/770.2 -> 78%

Parte 4. Efecto de las sentencias condicionales en el cuerpo del bucle

1. Lista el código ensamblador correspondiente al bucle interno de la función `cond_vec()`.
   ¿Cuántas instrucciones vectoriales hay en el cuerpo del bucle?

```
vmovaps 0x6020c0(%rax),%ymm2
vmovaps 0x6030c0(%rax),%ymm3
add    $0x20,%rax
vcmpltps %ymm1,%ymm2,%ymm0
vblendvps %ymm0,%ymm2,%ymm3,%ymm0
vmovaps %ymm0,0x6010a0(%rax)
cmp    $0x1000,%rax
jne    400940 <cond_vec+0x40>
```

8 instrucciones, de las cuales 5 son vectoriales.

2. Detalla las operaciones realizadas por las instrucciones vectoriales del bucle.
   Por ejemplo:

```
vmovaps 0x6020c0(%rax),%ymm2 -> lee de memoria 8 elementos del vector y[]
vmovaps 0x6030c0(%rax),%ymm3 -> lee de memoria 8 elementos del vector x[]
```

```
vcmpltps %ymm1,%ymm2,%ymm0 ->   genera una máscara comparando los 8 elementos
    leídos del vector y, si el elemento es menor al umbral escribe 1, else 0
vblendvps %ymm0,%ymm2,%ymm3,%ymm0 -> selecciona los elementos leídos de ambos
    vectores, si la máscara es 1 escribe elemento del vector y, else del x
vmovaps %ymm0,0x6010a0(%rax) -> escribe en memoria los 8 elementos
    resultantes de la instr anterior en el vector z[]
```

3. Calcula la aceleración (*speedup*) de la versión vectorial sobre la escalar.
   573.0/89.6 -> 639%