Práctica 5: Pralelización automática y mediante directivas OpenMP. Multiprocesadores

Autor: Héctor Lacueva Sacristán NIP: 869637

Fecha: 28/04/2025

Índice

Análisis de las dependencias del código

El siguiente trozo de código muestra el bucle principal del programa.

```
void main(int argc, char *argv){
  int nsubintervals; // numero de subintervalos en que se divide el intervalo [0,1]
  double subinterval, x;
  double area = 0.0;

  nsubinterval = atoi(argv[1]);
  subinterval =  1.0 / nsubintervals;

  for (int i = 0; i < nsubintervals; i++){
    x = (i-0.5)*subinterval;  // S1
    area = area + 4.0/(1.0 + x*x);   // S2
  }
  std::cout << "Valor de pi aproximado: " << area << std::endl;
}
```

Tras realizar el análisis de dependencias los resultados son los siguientes:

- Existe un **flow-dependency** de S1 a S2 a distancia 0.
- Existe una **anti-dependency** de S2 a S2 a distancia 0.
- Además de dependencias entre iteraciones para las variables `x` y `area`.

El código como está no es **ni vectorizable ni paralelizable**.

Para poder paralelizar `x` debería ser privada para todas las iteraciones y se debería de aplicar una operación de reducción suma sobre la variable `area`. De esta manera las dependencias no desaparecerían.

Ejecución secuencial

Experimento 1: Programa Serie

En el Apéndice 1.1 se puede encontrar el **trozo de código perteneciente al bucle principal compilado con -O0**. En el se puede ver que no se han aplicado optimizaciones y es un código bastante fácil de entender y no muy abultado.

En el Apéndice 1.2 se puede encontrar el **trozo de código perteneciente al bucle principal compilado con -O3**. De primeras se aprecia que el tamaño del códgigo es muchísimo mayor, principalmente por el desenrrollado de bucle que ha llevado a cabo el compilador. Esto hace que sea difícil de interpretar.

La siguiente tabla representa el tiempo de ejecucución del comando `./pi_serie 1000000000` con las diferentes opciones de compilación.

`-O0`	`-O3`
CPU clock = 7.11288 sg Wall Clock = 7.11299 sg MFLOPS = 843,54	CPU clock = 7.11780 sg Wall Clock = 7.11792 sg MFLOPS = 842,95

Como se puede ver en la tabla, la versión compilada con `-O0` es ligeramente más rápida con respecto a la versión compilada con `-O3`. Esto indica que las optimizaciones aplicadas con la opción `-O3` no han sido eficaces.

Experimento 2: Desenrollado de bucle

Falta calcular MFLOPS y speedup, aparte de añadir referencias apendice y comentar optimizaciones realizadas por el procesador.

`-O0`	`-O3`
CPU clock = 7.08547 sg Wall Clock = 7.08558 sg MFLOPS = 843,54	CPU clock = 6.92511 sg Wall Clock = 6.92646 sg MFLOPS = 842,95

Experimento 3: reducción del número de operaciones

Falta calcular MFLOPS y speedup, aparte de añadir referencias apendice y comentar optimizaciones realizadas por el procesador.

-O0	-O3
CPU clock = 7.08547 sg	CPU clock = 6.92511 sg
Wall Clock = 7.08558 sg	Wall Clock = 6.92646 sg
MFLOPS = 843,54	MFLOPS = 842,95

Apéndice 1

1. Experimento 1 -O0

El siguiente trozo de código hace referencia al apartado Experimento 1. En el se muestra el código generado por el compilador con -O0.

```
  for (i = 0; i < nsubintervals; i++){
  400d2c:   f9004bff    str xzr, [sp, #144]
  400d30:   14000014    b   400d80 <main+0x1cc>
    double x = (i-0.5)*subinterval;  // S1
  400d34:   fd404be0    ldr d0, [sp, #144]
  400d38:   5e61d801    scvtf   d1, d0
  400d3c:   1e6c1000    fmov    d0, #5.00000000000000000e-01
  400d40:   1e603820    fsub    d0, d1, d0
  400d44:   fd403fe1    ldr d1, [sp, #120]
  400d48:   1e600820    fmul    d0, d1, d0
  400d4c:   fd0033e0    str d0, [sp, #96]
    area = area + 4.0/(1.0 + x*x);   // S2
  400d50:   fd4033e0    ldr d0, [sp, #96]
  400d54:   1e600801    fmul    d1, d0, d0
  400d58:   1e6e1000    fmov    d0, #1.00000000000000000e+00
  400d5c:   1e602820    fadd    d0, d1, d0
  400d60:   1e621001    fmov    d1, #4.00000000000000000e+00
  400d64:   1e601820    fdiv    d0, d1, d0
  400d68:   fd404fe1    ldr d1, [sp, #152]
  400d6c:   1e602820    fadd    d0, d1, d0
  400d70:   fd004fe0    str d0, [sp, #152]
  for (i = 0; i < nsubintervals; i++){
  400d74:   f9404be0    ldr x0, [sp, #144]
  400d78:   91000400    add x0, x0, #0x1
  400d7c:   f9004be0    str x0, [sp, #144]
  400d80:   f9404be1    ldr x1, [sp, #144]
  400d84:   f94047e0    ldr x0, [sp, #136]
  400d88:   eb00003f    cmp x1, x0
  400d8c:   54fffd4b    b.lt    400d34 <main+0x180>  // b.tstop
  }
```

2. Experimento 1 -O3

El siguiente trozo de código hace referencia al apartado Experimento 1. En el se muestra el código generado por el compilador con -O3.

```
for (i = 0; i < nsubintervals; i++){
  400be4:   f100029f    cmp x20, #0x0
  400be8:   5400184d    b.le    400ef0 <main+0x440>
  400bec:   d1000680    sub x0, x20, #0x1
  400bf0:   f100181f    cmp x0, #0x6
  400bf4:   54001829    b.ls    400ef8 <main+0x448>  // b.plast
  400bf8:   b0000000    adrp    x0, 401000 <register_tm_clones+0x20>
  area = 0.0;
  400bfc:   2f00e403    movi    d3, #0x0
```

```
400c00:   4e080409    dup v9.2d, v0.d[0]
400c04:   d341fe82    lsr x2, x20, #1
for (i = 0; i < nsubintervals; i++){
400c08:   3dc0bc04    ldr q4, [x0, #752]
400c0c:   b0000000    adrp    x0, 401000 <register_tm_clones+0x20>
  double x = (i-0.5)*subinterval;  // S1
400c10:   6f07f407    fmov    v7.2d, #-5.00000000000000000e-01
for (i = 0; i < nsubintervals; i++){
400c14:   d2800001    mov x1, #0x0                    // #0
400c18:   3dc0c008    ldr q8, [x0, #768]
  area = area + 4.0/(1.0 + x*x);   // S2
400c1c:   6f03f606    fmov    v6.2d, #1.00000000000000000e+00
400c20:   6f00f605    fmov    v5.2d, #4.00000000000000000e+00
400c24:   d503201f    nop
400c28:   4ea41c81    mov v1.16b, v4.16b
400c2c:   91000421    add x1, x1, #0x1
400c30:   4ea61cc2    mov v2.16b, v6.16b
400c34:   4ee88484    add v4.2d, v4.2d, v8.2d
  double x = (i-0.5)*subinterval;  // S1
400c38:   4e61d821    scvtf   v1.2d, v1.2d
400c3c:   4e67d421    fadd    v1.2d, v1.2d, v7.2d
400c40:   6e69dc21    fmul    v1.2d, v1.2d, v9.2d
  area = area + 4.0/(1.0 + x*x);   // S2
400c44:   4e61cc22    fmla    v2.2d, v1.2d, v1.2d
400c48:   6e62fca1    fdiv    v1.2d, v5.2d, v2.2d
400c4c:   1e604022    fmov    d2, d1
400c50:   5e180421    mov d1, v1.d[1]
400c54:   1e622862    fadd    d2, d3, d2
400c58:   1e622823    fadd    d3, d1, d2
for (i = 0; i < nsubintervals; i++){
400c5c:   eb01005f    cmp x2, x1
400c60:   54fffe41    b.ne    400c28 <main+0x178>  // b.any
400c64:   927ffa81    and x1, x20, #0xfffffffffffffffe
400c68:   36000814    tbz w20, #0, 400d68 <main+0x2b8>
  double x = (i-0.5)*subinterval;  // S1
400c6c:   9e620021    scvtf   d1, x1
400c70:   1e6c1005    fmov    d5, #5.00000000000000000e-01
  area = area + 4.0/(1.0 + x*x);   // S2
400c74:   1e6e1004    fmov    d4, #1.00000000000000000e+00
400c78:   1e621002    fmov    d2, #4.00000000000000000e+00
for (i = 0; i < nsubintervals; i++){
400c7c:   91000422    add x2, x1, #0x1
  double x = (i-0.5)*subinterval;  // S1
400c80:   1e653821    fsub    d1, d1, d5
400c84:   1e600821    fmul    d1, d1, d0
  area = area + 4.0/(1.0 + x*x);   // S2
400c88:   1f411021    fmadd   d1, d1, d1, d4
400c8c:   1e611841    fdiv    d1, d2, d1
400c90:   1e612863    fadd    d3, d3, d1
for (i = 0; i < nsubintervals; i++){
400c94:   eb02029f    cmp x20, x2
400c98:   5400068d    b.le    400d68 <main+0x2b8>
  double x = (i-0.5)*subinterval;  // S1
400c9c:   9e620041    scvtf   d1, x2
for (i = 0; i < nsubintervals; i++){
400ca0:   91000822    add x2, x1, #0x2
  double x = (i-0.5)*subinterval;  // S1
400ca4:   1e653821    fsub    d1, d1, d5
400ca8:   1e600821    fmul    d1, d1, d0
  area = area + 4.0/(1.0 + x*x);   // S2
400cac:   1f411021    fmadd   d1, d1, d1, d4
400cb0:   1e611841    fdiv    d1, d2, d1
```

```
400cb4:   1e612863    fadd    d3, d3, d1
for (i = 0; i < nsubintervals; i++){
400cb8:   eb02029f    cmp x20, x2
400cbc:   5400056d    b.le    400d68 <main+0x2b8>
  double x = (i-0.5)*subinterval;  // S1
400cc0:   9e620041    scvtf   d1, x2
for (i = 0; i < nsubintervals; i++){
400cc4:   91000c22    add x2, x1, #0x3
  double x = (i-0.5)*subinterval;  // S1
400cc8:   1e653821    fsub    d1, d1, d5
400ccc:   1e600821    fmul    d1, d1, d0
  area = area + 4.0/(1.0 + x*x);   // S2
400cd0:   1f411021    fmadd   d1, d1, d1, d4
400cd4:   1e611841    fdiv    d1, d2, d1
400cd8:   1e612863    fadd    d3, d3, d1
for (i = 0; i < nsubintervals; i++){
400cdc:   eb02029f    cmp x20, x2
400ce0:   5400044d    b.le    400d68 <main+0x2b8>
  double x = (i-0.5)*subinterval;  // S1
400ce4:   9e620041    scvtf   d1, x2
for (i = 0; i < nsubintervals; i++){
400ce8:   91001022    add x2, x1, #0x4
  double x = (i-0.5)*subinterval;  // S1
400cec:   1e653821    fsub    d1, d1, d5
400cf0:   1e600821    fmul    d1, d1, d0
  area = area + 4.0/(1.0 + x*x);   // S2
400cf4:   1f411021    fmadd   d1, d1, d1, d4
400cf8:   1e611841    fdiv    d1, d2, d1
400cfc:   1e612863    fadd    d3, d3, d1
for (i = 0; i < nsubintervals; i++){
400d00:   eb02029f    cmp x20, x2
400d04:   5400032d    b.le    400d68 <main+0x2b8>
  double x = (i-0.5)*subinterval;  // S1
400d08:   9e620041    scvtf   d1, x2
for (i = 0; i < nsubintervals; i++){
400d0c:   91001422    add x2, x1, #0x5
  double x = (i-0.5)*subinterval;  // S1
400d10:   1e653821    fsub    d1, d1, d5
400d14:   1e600821    fmul    d1, d1, d0
  area = area + 4.0/(1.0 + x*x);   // S2
400d18:   1f411021    fmadd   d1, d1, d1, d4
400d1c:   1e611841    fdiv    d1, d2, d1
400d20:   1e612863    fadd    d3, d3, d1
for (i = 0; i < nsubintervals; i++){
400d24:   eb02029f    cmp x20, x2
400d28:   5400020d    b.le    400d68 <main+0x2b8>
  double x = (i-0.5)*subinterval;  // S1
400d2c:   9e620041    scvtf   d1, x2
for (i = 0; i < nsubintervals; i++){
400d30:   91001821    add x1, x1, #0x6
  double x = (i-0.5)*subinterval;  // S1
400d34:   1e653821    fsub    d1, d1, d5
400d38:   1e600821    fmul    d1, d1, d0
  area = area + 4.0/(1.0 + x*x);   // S2
400d3c:   1f411021    fmadd   d1, d1, d1, d4
400d40:   1e611841    fdiv    d1, d2, d1
400d44:   1e612863    fadd    d3, d3, d1
for (i = 0; i < nsubintervals; i++){
400d48:   eb01029f    cmp x20, x1
400d4c:   540000ed    b.le    400d68 <main+0x2b8>
  double x = (i-0.5)*subinterval;  // S1
400d50:   9e620021    scvtf   d1, x1
```

```
  400d54:   1e653821    fsub    d1, d1, d5
  400d58:   1e600821    fmul    d1, d1, d0
    area = area + 4.0/(1.0 + x*x);   // S2
  400d5c:   1f411021    fmadd   d1, d1, d1, d4
  400d60:   1e611842    fdiv    d2, d2, d1
  400d64:   1e622863    fadd    d3, d3, d2
  }
```

3. Experimento 2 -O0

El siguiente trozo de código hace referencia al apartado Experimento 2. En el se muestra el código generado por el compilador con -O0.

```
for (i = 0; i < nsubintervals; i+=4){
  400d48:   f90057ff    str xzr, [sp, #168]
  400d4c:   1400004c    b   400e7c <main+0x2c8>
    x = (i-0.5)*subinterval;
  400d50:   fd4057e0    ldr d0, [sp, #168]
  400d54:   5e61d801    scvtf   d1, d0
  400d58:   1e6c1000    fmov    d0, #5.00000000000000000e-01
  400d5c:   1e603820    fsub    d0, d1, d0
  400d60:   fd404be1    ldr d1, [sp, #144]
  400d64:   1e600820    fmul    d0, d1, d0
  400d68:   fd003fe0    str d0, [sp, #120]
    varea[0] = varea[0] + 4.0/(1.0 + x*x);
  400d6c:   910143e0    add x0, sp, #0x50
  400d70:   fd400001    ldr d1, [x0]
  400d74:   fd403fe0    ldr d0, [sp, #120]
  400d78:   1e600802    fmul    d2, d0, d0
  400d7c:   1e6e1000    fmov    d0, #1.00000000000000000e+00
  400d80:   1e602840    fadd    d0, d2, d0
  400d84:   1e621002    fmov    d2, #4.00000000000000000e+00
  400d88:   1e601840    fdiv    d0, d2, d0
  400d8c:   1e602820    fadd    d0, d1, d0
  400d90:   910143e0    add x0, sp, #0x50
  400d94:   fd000000    str d0, [x0]
    x = (i+0.5)*subinterval;
  400d98:   fd4057e0    ldr d0, [sp, #168]
  400d9c:   5e61d801    scvtf   d1, d0
  400da0:   1e6c1000    fmov    d0, #5.00000000000000000e-01
  400da4:   1e602820    fadd    d0, d1, d0
  400da8:   fd404be1    ldr d1, [sp, #144]
  400dac:   1e600820    fmul    d0, d1, d0
  400db0:   fd003fe0    str d0, [sp, #120]
    varea[1] = varea[1] + 4.0/(1.0 + x*x);
  400db4:   910143e0    add x0, sp, #0x50
  400db8:   fd400401    ldr d1, [x0, #8]
  400dbc:   fd403fe0    ldr d0, [sp, #120]
  400dc0:   1e600802    fmul    d2, d0, d0
  400dc4:   1e6e1000    fmov    d0, #1.00000000000000000e+00
  400dc8:   1e602840    fadd    d0, d2, d0
  400dcc:   1e621002    fmov    d2, #4.00000000000000000e+00
  400dd0:   1e601840    fdiv    d0, d2, d0
  400dd4:   1e602820    fadd    d0, d1, d0
  400dd8:   910143e0    add x0, sp, #0x50
  400ddc:   fd000400    str d0, [x0, #8]
    x = (i+1.5)*subinterval;
  400de0:   fd4057e0    ldr d0, [sp, #168]
  400de4:   5e61d801    scvtf   d1, d0
  400de8:   1e6f1000    fmov    d0, #1.50000000000000000e+00
  400dec:   1e602820    fadd    d0, d1, d0
  400df0:   fd404be1    ldr d1, [sp, #144]
```

```
  400df4:   1e600820    fmul    d0, d1, d0
  400df8:   fd003fe0    str d0, [sp, #120]
    varea[2] = varea[2] + 4.0/(1.0 + x*x);
  400dfc:   910143e0    add x0, sp, #0x50
  400e00:   fd400801    ldr d1, [x0, #16]
  400e04:   fd403fe0    ldr d0, [sp, #120]
  400e08:   1e600802    fmul    d2, d0, d0
  400e0c:   1e6e1000    fmov    d0, #1.0000000000000000e+00
  400e10:   1e602840    fadd    d0, d2, d0
  400e14:   1e621002    fmov    d2, #4.0000000000000000e+00
  400e18:   1e601840    fdiv    d0, d2, d0
  400e1c:   1e602820    fadd    d0, d1, d0
  400e20:   910143e0    add x0, sp, #0x50
  400e24:   fd000800    str d0, [x0, #16]
    x = (i+2.5)*subinterval;
  400e28:   fd4057e0    ldr d0, [sp, #168]
  400e2c:   5e61d801    scvtf   d1, d0
  400e30:   1e609000    fmov    d0, #2.5000000000000000e+00
  400e34:   1e602820    fadd    d0, d1, d0
  400e38:   fd404be1    ldr d1, [sp, #144]
  400e3c:   1e600820    fmul    d0, d1, d0
  400e40:   fd003fe0    str d0, [sp, #120]
    varea[3] = varea[3] + 4.0/(1.0 + x*x);
  400e44:   910143e0    add x0, sp, #0x50
  400e48:   fd400c01    ldr d1, [x0, #24]
  400e4c:   fd403fe0    ldr d0, [sp, #120]
  400e50:   1e600802    fmul    d2, d0, d0
  400e54:   1e6e1000    fmov    d0, #1.0000000000000000e+00
  400e58:   1e602840    fadd    d0, d2, d0
  400e5c:   1e621002    fmov    d2, #4.0000000000000000e+00
  400e60:   1e601840    fdiv    d0, d2, d0
  400e64:   1e602820    fadd    d0, d1, d0
  400e68:   910143e0    add x0, sp, #0x50
  400e6c:   fd000c00    str d0, [x0, #24]
  for (i = 0; i < nsubintervals; i+=4){
  400e70:   f94057e0    ldr x0, [sp, #168]
  400e74:   91001000    add x0, x0, #0x4
  400e78:   f90057e0    str x0, [sp, #168]
  400e7c:   f94057e1    ldr x1, [sp, #168]
  400e80:   f94053e0    ldr x0, [sp, #160]
  400e84:   eb00003f    cmp x1, x0
  400e88:   54fff64b    b.lt    400d50 <main+0x19c>  // b.tstop
}
```

4. Experimento 2 -O3

El siguiente trozo de código hace referencia al apartado Experimento 2. En el se muestra el código generado por el compilador con -O3.

```
for (i = 0; i < nsubintervals; i+=4){
  400be4:   f100029f    cmp x20, #0x0
  400be8:   540011cd    b.le    400e20 <main+0x370>
    varea[0] = varea[0] + 4.0/(1.0 + x*x);
    x = (i+0.5)*subinterval;
    varea[1] = varea[1] + 4.0/(1.0 + x*x);
    x = (i+1.5)*subinterval;
    varea[2] = varea[2] + 4.0/(1.0 + x*x);
    x = (i+2.5)*subinterval;
  400bec:   b0000000    adrp    x0, 401000 <__libc_csu_init>
  400bf0:   d1000682    sub x2, x20, #0x1
  for (i = 0; i < nsubintervals; i+=4){
  400bf4:   6f00e404    movi    v4.2d, #0x0
```

```
400bf8:   d2800001    mov x1, #0x0                    // #0
  x = (i+2.5)*subinterval;
400bfc:   3dc08812    ldr q18, [x0, #544]
400c00:   b0000000    adrp    x0, 401000 <__libc_csu_init>
400c04:   4e080668    dup v8.2d, v19.d[0]
400c08:   d342fc42    lsr x2, x2, #2
for (i = 0; i < nsubintervals; i+=4){
400c0c:   4f000400    movi    v0.4s, #0x0
400c10:   91000442    add x2, x2, #0x1
400c14:   4ea41c85    mov v5.16b, v4.16b
  x = (i+2.5)*subinterval;
400c18:   6f03f411    fmov    v17.2d, #5.0000000000000000e-01
  x = (i-0.5)*subinterval;
400c1c:   6f07f410    fmov    v16.2d, #-5.0000000000000000e-01
  varea[3] = varea[3] + 4.0/(1.0 + x*x);
400c20:   6f03f607    fmov    v7.2d, #1.0000000000000000e+00
400c24:   6f00f606    fmov    v6.2d, #4.0000000000000000e+00
400c28:   3dc08c09    ldr q9, [x0, #560]
400c2c:   d503201f    nop
400c30:   91000421    add x1, x1, #0x1
  x = (i-0.5)*subinterval;
400c34:   4e61d801    scvtf   v1.2d, v0.2d
400c38:   4ee98400    add v0.2d, v0.2d, v9.2d
400c3c:   4e70d423    fadd    v3.2d, v1.2d, v16.2d
  x = (i+2.5)*subinterval;
400c40:   4e71d422    fadd    v2.2d, v1.2d, v17.2d
400c44:   4e72d421    fadd    v1.2d, v1.2d, v18.2d
400c48:   6e184462    mov v2.d[1], v3.d[1]
400c4c:   6e68dc21    fmul    v1.2d, v1.2d, v8.2d
  varea[3] = varea[3] + 4.0/(1.0 + x*x);
400c50:   4ea71ce3    mov v3.16b, v7.16b
  x = (i+2.5)*subinterval;
400c54:   6e68dc42    fmul    v2.2d, v2.2d, v8.2d
  varea[3] = varea[3] + 4.0/(1.0 + x*x);
400c58:   4e61cc23    fmla    v3.2d, v1.2d, v1.2d
400c5c:   4ea71ce1    mov v1.16b, v7.16b
400c60:   4e62cc41    fmla    v1.2d, v2.2d, v2.2d
400c64:   6e63fcc2    fdiv    v2.2d, v6.2d, v3.2d
400c68:   6e61fcc1    fdiv    v1.2d, v6.2d, v1.2d
400c6c:   4e62d4a5    fadd    v5.2d, v5.2d, v2.2d
400c70:   4e61d484    fadd    v4.2d, v4.2d, v1.2d
400c74:   eb01005f    cmp x2, x1
400c78:   54fffdc8    b.hi    400c30 <main+0x180>  // b.pmore
400c7c:   1e6040a0    fmov    d0, d5
400c80:   1e604081    fmov    d1, d4
400c84:   5e1804a5    mov d5, v5.d[1]
400c88:   5e180488    mov d8, v4.d[1]
}
```