

Examen-BP4-Resuelto-Zukii.pdf



Zukii



Arquitectura de Computadores



2º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

EMMA STONE

Disney

Cruella

EN
CINES

O DISFRÚTALA
EN

Disney+

A TRAVÉS DE
ACCESO PREMIUM
Con coste adicional

©2021 Disney Enterprises, Inc. Requiere de una suscripción activa a Disney+ y un único pago adicional por el Acceso Premium a Cruella. Disfrútala en aquellos cines donde esté disponible y con Acceso Premium antes de que esté disponible para todos los suscriptores a Disney+.



@Zukii on Wuolah

Examen BP4 Resuelto.

Recuerda que me puedes seguir en wuolah :P

Pd: para cuando subo el examen, no se la nota ni los resultados, así que puede ser que alguna respuesta no sea correcta, aunque están consensuadas post-examen. Y otra cosa, no te fijas en las respuestas marcadas, fíjate en las respuestas.

1) ¿Cuál de los Sigüientes códigos es computacionalmente más eficiente?

¿Cuál de los siguientes códigos es computacionalmente más eficiente?

Usaria Profesores

- ☐ a) `x = w & 7;`
`y = x * x;`
`z = (y << 5)+y;`
`for (i = h = 0 ; i < MAX ; i++) {`
`h += 14;`
`}`
- ☐ b) `x = w % 8;`
`y = pow(x, 2.0);`
`z = y * 33;`
`for (i = 0 ; i < MAX ; i++) {`
`h = 14 * i;`
`}`
- ☐ c) `x = w & 7;`
`y = pow (x, 2,0);`
`z = (y << 5)+y;`
`for (i = h = 0 ; i < MAX ; i++) {`
`h += 14;`
`}`
- ☐ d) `x = w % 8;`
`y = x * x;`
`z = (y << 5)+y;`
`for (i = 0 ; i < MAX ; i++) {`
`h = 14 * i;`
`}`

Respuesta: a) ya que no hace uso de la función pow y hace sumas, que son más eficientes que el producto.

2) ¿A que función de C podría corresponder el siguiente código ensamblador?

```
0x4005d0 <+0>:  cmp    %esi, %edi
0x4005d2 <+2>:  mov     %esi, %eax
0x4005d4 <+4>:  cmovle %edi, %eax
0x4005d7 <+7>:  retq
```

Usuaría Profesores

- ☒ a)

```
int f(int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```
- ☒ b)

```
int f(int a, int b) {
    if (a < b)
        return a;
    else
        return b;
}
```
- ☒ c) ninguna otra respuesta es correcta
- ☒ d)

```
int f(int a, int b, int c, int d) {
    if (a < b)
        return c;
    else
        return d;
}
```

Respuesta: a) ya que cmovle hace salto if lower of equal (la parte del else)

3) ¿Cuál de las siguientes ventajas aporta la técnica de desenrollado de bucle?


- ☒ a) elimina todas las instrucciones de salto del bucle
- ☒ b) ninguna otra respuesta es correcta
- ☒ c) aumenta la velocidad de ejecución
- ☒ d) reduce el tamaño del código


Respuesta: c) como se puede ver en las transparencias de la práctica.


- 4) ¿Cuál de las optimizaciones siguientes reduce el tiempo de ejecución del código que se muestra a continuación? (M y N son múltiplos de dos).


```
for (i=0; i<M; i++)
  for (j=0; j<N; j++){
    if ((j%2)==0)
      c[j] += a[j][i]+b[j][i];
    else
      c[j] += a[j][i]-b[j][i];
  }
```

Usaria Profesores

-  a)

```
for (i=0; i<M; i++)
  for (j=0; j<N; j++){
    c[j] += a[j][i]+b[j][i];
    c[j+1] += a[j+1][i]-b[j+1][i];
  }
```
-  b)

```
for (j=0; j<M; j++)
  for (i=0; i<N; i++){
    if ((i%2)==0)
      c[i] += a[i][j]+b[i][j];
    else
      c[i] += a[i][j]-b[i][j];
  }
```
-  c)

```
for (i=0; i<M; i++)
  for (j=0; j<N; j+=2){
    c[j] += a[j][i]+b[j][i];
    c[j+1] += a[j+1][i]-b[j+1][i];
  }
```
-  d)

```
for (i=0; i<M; i+=2)
  for (j=0; j<N; j+=2){
    c[j] += a[j][i]+b[j][i];
    c[j+1] += a[j+1][i+1]-b[j+1][i+1];
  }
```

Respuesta: c) ya que es correcto el resultado y hacemos el desenrollado del bucle que se hace con j

- 5) Dado el siguiente código y suponiendo el vector v inicializado, ¿qué opción es verdadera?

```
for (int i = 0; i < 1000; ++i)
{
    if ((v[i] % 3) == 0)
        foo(v[i]);
    else
        switch((v[i] % 3))
        {
            case 1: foo(v[i] + 2); break;
            case 2: foo(v[i] + 1); break;
        }
}
```

Usaria Profesores

- ☐ a) sólo el desenrollado de bucle puede servir para optimizar el código
- ☐ b) los valores contenidos en v no afectan a la velocidad de ejecución
- ☐ c) la ejecución finaliza antes si v contiene muchos múltiplos de 3
- ☐ d) la ejecución finaliza antes si v no contiene ningún múltiplo de 3

Respuesta: la c), ya que el código irá mejor cuando son múltiplos de 3. Por eso se ha sacado del switch-case.

- 6) Sin indicarle un núcleo concreto, ¿cómo ordenaría las instrucciones con enteros en orden creciente de tiempo de ejecución?

- ☐ a) Multiplicación, división y desplazamiento de bits
- ☐ b) División, desplazamiento de bits y multiplicación
- ☐ c) Desplazamientos de bits, multiplicación y división
- ☐ d) Desplazamiento de bits, división y multiplicación

Respuesta: c) Ya que desplazar es más eficiente que multiplicar, y a su vez, multiplicar que dividir.

@Zukii on Wuolah

7) ¿Cuál de las siguientes afirmaciones es correcta?

- ☐ a) el proceso de optimización se debe realizar siempre al final del desarrollo de la aplicación
- ☐ b) ninguna otra respuesta es correcta
- ☐ c) hay optimizaciones que son aplicables a cualquier procesador
- ☐ d) la optimización de código siempre debe realizarse en lenguaje ensamblador

Respuesta: c) como se puede ver en unas de las primeras transparencias del tema.

8) ¿Cómo cree que implementará el compilador una función que multiplica un entero por 11 al compilar con optimización máxima (-O3)?

- ```
int function_f(int x)
{
 return x * 11;
}
```
- a) 0x401116 <+0>: imul \$0xb,%edi,%eax  
0x401119 <+3>: retq
  - b) 0x401120 <+0>: lea (%rdi,%rdi,4),%eax  
0x401123 <+3>: lea (%rdi,%rax,2),%eax  
0x401126 <+6>: retq
  - c) 0x401106 <+0>: push %rbp  
0x401107 <+1>: mov %rsp,%rbp  
0x40110a <+4>: mov %edi,-0x4(%rbp)  
0x40110d <+7>: mov -0x4(%rbp),%edx  
0x401110 <+10>: mov %edx,%eax  
0x401112 <+12>: shl \$0x2,%eax  
0x401115 <+15>: add %edx,%eax  
0x401117 <+17>: add %eax,%eax  
0x401119 <+19>: add %edx,%eax  
0x40111b <+21>: pop %rbp  
0x40111c <+22>: retq
  - d) ninguna otra respuesta es correcta

Respuesta: c), ya que -O3 es la que más instrucciones en ensamblador genera.

- 9) ¿Qué código cree que calculará de forma correcta y en menor tiempo el producto de dos matrices en un sistema multiprocesador? Suponga matrices cuadradas, c inicializada a cero y N muy grande

```
int a[N][N], b[N][N], c[N][N];
a) for (int i=0; i<N; ++i)
 #pragma omp parallel for
 for (int j=0; j<N; ++j)
 for (int k=0; k<N; ++k)
 c[i][j] += a[i][k] * b[k][j];
b) for (int i=0; i<N; ++i)
 #pragma omp parallel for
 for (int j=0; j<N; ++j)
 for (int k=0; k<N; ++k)
 #pragma omp atomic
 c[i][j] += a[i][k] * b[k][j];
c) for (int i=0; i<N; ++i)
 #pragma omp parallel for
 for (int j=0; j<N; ++j)
 for (int k=0; k<N; ++k)
 #pragma omp critical
 c[i][j] += a[i][k] * b[k][j];
d) for (int i=0; i<N; ++i)
 for (int j=0; j<N; ++j)
 for (int k=0; k<N; ++k)
 c[i][j] += a[i][k] * b[k][j];
```

Respuesta: b) ya que hacemos el producto de forma paralela, y como hay una SC que es sumar lo que llevemos calculador a  $c[i][j]$ , debemos usar o bien la directiva atomic o critical. Es mejor atomic, porque critical solo puede acceder la hebra master.



10) ¿Cuál de las siguientes formas de implementar el algoritmo cree más rápida?

```
const int N = 5000, REP = 40000;
int R[REP + 1];
struct S { int a, b; } s[N];
a) int sa = 0, sb = 0;
 for (int i = 0; i < N; ++i)
 {
 sa += s[i].a;
 sb += s[i].b;
 }
 sa *= 2;
 sb *= 3;
 for (int ii = 1; ii <= REP; ++ii)
 R[ii] = std::min(sa + N * ii, sb - N * ii);
b) struct { int x1, x2; } x[N];

 for (int i = 0; i < N; ++i)
 {
 x[i].x1 = 2 * s[i].a;
 x[i].x2 = 3 * s[i].b;
 }

 for (int ii = 1; ii <= REP; ++ii)
 {
 int x1 = 0, x2 = 0;
 for (int i = 0; i < N; ++i)
 {
 x1 += x[i].x1 + ii;
 x2 += x[i].x2 - ii;
 }
 R[ii] = std::min(x1, x2);
 }
c) for (int ii = 1; ii <= REP; ++ii)
 {
 int x1 = 0, x2 = 0;
 for (int i = 0; i < N; ++i)
 {
 x1 += 2 * s[i].a + ii;
 x2 += 3 * s[i].b - ii;
 }
 R[ii] = std::min(x1, x2);
 }
d) for (int ii = 1; ii <= REP; ++ii)
 {
 int X1 = 0, X2 = 0;
 for (int i = 0; i < N; ++i)
 X1 += 2 * s[i].a + ii;
 for (int i = 0; i < N; ++i)
 X2 += 3 * s[i].b - ii;
 if (X1 < X2)
 R[ii] = X1;
 else
 R[ii] = X2;
 }
```

Respuesta a), ya que no tiene bucles anidados.

11) ¿Cómo cree que se calcularía más rápido la operación "a = b \* c" suponiendo que el valor de c es 5?

Valor de c es 5.

- a) a = b + b + b + b + b;
- b) a = b + (b << 2);
- c) a = b \* c;
- d) a = c \* b;

Respuesta: b) ya que desplazamiento es mejor que sumar 5 veces.

12) ¿Cómo ordenaría los índices del siguiente algoritmo de multiplicación de matrices?

```
int a[100][100], b[100][100], c[100][100];
```

Usuario Profesores

- ☐ a) for (int i = 0; i < 100; ++i)  
for (int j = 0; j < 100; ++j)  
for (int k = 0; k < 100; ++k)  
a[i][j] += b[i][k] \* c[k][j];
- ☐ b) for (int i = 0; i < 100; ++i)  
for (int k = 0; k < 100; ++k)  
for (int j = 0; j < 100; ++j)  
a[i][j] += b[i][k] \* c[k][j];
- ☐ c) for (int j = 0; j < 100; ++j)  
for (int i = 0; i < 100; ++i)  
for (int k = 0; k < 100; ++k)  
a[i][j] += b[i][k] \* c[k][j];
- ☐ d) for (int k = 0; k < 100; ++k)  
for (int j = 0; j < 100; ++j)  
for (int i = 0; i < 100; ++i)  
a[i][j] += b[i][k] \* c[k][j];

Respuesta: b), ya que como vimos en un ejercicio de la BP4, conviene cambiar k por j para aprovechar el acceso a memoria por columnas.

@Zukii on Wuolah

- 13) ¿Cuál de las siguientes versiones de una función que multiplica un entero por 6 cree que se obtendrá al compilar con optimización en espacio (-Os)?

```
int f(int x)
{
 return x * 6;
}
```

Usuario Profesores

- ☐ a) 0x401106 <+0>: lea (%rdi,%rdi,2),%eax  
0x401109 <+3>: add %eax,%eax  
0x40110b <+5>: retq
- ☐ b) 0x401116 <+0>: imul \$0x6,%edi,%eax  
0x401119 <+3>: retq
- ☐ c) 0x401106 <+0>: push %rbp  
0x401107 <+1>: mov %rsp,%rbp  
0x40110a <+4>: mov %edi,-0x4(%rbp)  
0x40110d <+7>: mov -0x4(%rbp),%edx  
0x401110 <+10>: mov %edx,%eax  
0x401112 <+12>: add %eax,%eax  
0x401114 <+14>: add %edx,%eax  
0x401116 <+16>: add %eax,%eax  
0x401118 <+18>: pop %rbp  
0x401119 <+19>: retq
- ☐ d) ninguna otra respuesta es correcta

Respuesta: b) ya que es la que menos número de instrucciones tiene.



14) ¿Cuál de las siguientes alternativas es mejor para ... (no la copio entera que me da un jamacuco como lo haga)

¿Cuál de las siguientes alternativas es mejor para implementar  $ebx = (A < B)$ ?  
C1: C2 teniendo en cuenta que: se repite su ejecución en el programa 100 veces, todas las instrucciones suponen siempre 1 ciclo (entonces el Código 2 supone siempre 6 ciclos y el Código 1 supone siempre 4 ciclos si no se salta en la instrucción de salto condicional), la predicción del salto condicional es siempre No Saltar, la predicción falla 2 de cada 10 ejecuciones del salto condicional y suponiendo que la penalización por predicción incorrecta es siempre 8 ciclos?

Código 1:  

```
cmpl %ebx, %eax
jge .L1
movl %ecx, %ebx // ecx=C1
jmp .L2
.L1:
movl %edx, %ebx // edx=C2
.L2:
```

Código 2:  

```
xorl %ebx, %ebx
cmpl %ebx, %eax
setlge %bl
decl %ebx
andl %ecx, %ebx // ecx=C1-C2
addl %edx, %ebx // edx=C2
```

- a) Código 2 en este caso
- b) Hay un empate entre los dos
- c) Código 1 en este caso
- d) Falta información

Respuesta: c) ya que el código 2 hace 600 pero el código 1, hace  $4 \cdot 100$  ciclos de normal +  $8 \cdot 20$  cuando se equivoca, que como se equivoca 2 veces de 10, sería 560.

15) Indique qué opción se ejecutará más rápido dados

```
const int n = 1000000;
int a[n], b[n];
Usuario Profesores
a) for (i=0 ; i<n ; i++) {
 *p = *p + a[i]*b[i];
}
b) int tmp0=0, tmp1=0, tmp2=0, tmp3=0;
 for (i=0 ; i<n ; i+=4) {
 tmp0 += a[i] * b[i];
 tmp1 += a[i+1] * b[i+1];
 tmp2 += a[i+2] * b[i+2];
 tmp3 += a[i+3] * b[i+3];
 }
 *p = tmp0 + tmp1 + tmp2 + tmp3;
c) for (i=0 ; i<n ; i+=4) {
 *p += a[i] * b[i];
 *p += a[i+1] * b[i+1];
 *p += a[i+2] * b[i+2];
 *p += a[i+3] * b[i+3];
}
d) for (i=0 ; i<n ; ++i) {
 *p += a[i]*b[i];
}
```

Respuesta: b) como vimos en las transparencias.

16) ¿Cuál cree que es la implementación óptima del siguiente algoritmo?

```
int f(int n)
{
 int s = 0;
 for (int i = 0; i < n; ++i)
 s += i % 5 + 1;
 return s;
}
Usuario Profesores
a) ninguna otra respuesta es correcta
b) f(int):
 leal 0(,%rdi,4), %eax
 ret
c) f(int):
 xorl %ecx, %ecx
 xorl %r8d, %r8d
 movl $5, %esi
.L3:
 cmpl %edi, %ecx
 jge .L1
 movl %ecx, %eax
 incl %ecx
 cltd
 idivl %esi
 leal 1(%r8,%rdx), %r8d
 jmp .L3
.L1:
 movl %r8d, %eax
 ret
d) f(int):
 leal (%rdi,%rdi,2), %eax
 ret
```

Respuesta: c) ya que ni b) ni d) dan el resultado correcto. (Puedes probar manualmente el ejemplo de n=11)

Puede que me haya dejado preguntas que ahora mismo no encuentro, pero la dinámica es la similar.

Cabe resaltar que hay una sobre una secuencia de caracteres WSU donde (creo) que la más eficiente es tratar la secuencia como un vector y acceder a la componente, más que una construcción condicional.

Pues con estas sesiones y un bizcocho, a ver si se saca el 8.  
(maldita carrera, me está dejando loco, no acabéis como yo haciendo esto muerto de calor un viernes a las 17:44 :D).

Recuerda que subo más apuntes a mi cuenta por si me querés seguir <3.

Zukii