



Examen BP4 - Grupo 3



Universidad de Granada - Doble Grado en Ingeniería Informática y Matemáticas
Arquitectura de Computadores



Desconocido: 45338112 Guerrero Cano, Valentín



Inicio: Hoy, miércoles, 09:40:14

Final: Hoy, miércoles, 09:57:42

Preguntas: 10

Respuestas

válidas:

Puntuación:

Nota:

1
Elección única

¿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) 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
- c) ninguna otra respuesta es correcta
- d) 0x401116 <+0>: imul \$0x6,%edi,%eax
0x401119 <+3>: retq

2
Elección única

¿Cuál es la principal diferencia entre las opciones de optimización -O2 y -O3?

Usuario Profesores

- a) -O3 crea un binario más pequeño que -O2 a costa de perder un poco de rendimiento
- b) -O3 crea un binario que tendrá una ejecución más eficiente que -O2, aunque ambos binarios pueden ser ejecutados en máquinas diferentes de las que los crearon

-  c) -02 crea un binario que puede ser usado en cualquier modelo de CPU, mientras que -03 lo crea en función del que compila el código
-  d) -03 solo funciona para procesadores Intel, mientras que -02 puede usara para cualquier tipo de procesadores


3


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


Elección única


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

Usuario Profesores

-  a)

```
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];
```
-  b)

```
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];
```
-  c)

```
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];
```
-  d)

```
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];
```

4


Indique qué opción se ejecutará más rápido dados


Elección única


```
const int n = 1000000;
```


```
int a[n], b[n];
```

Usuario Profesores

-  a)

```
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];
}
```
-  b)

```
for (i=0 ; i<n ; ++i) {
  *p += a[i]*b[i];
}
```
-  c)

```
for (i=0 ; i<n ; i++) {
  *p = *p + a[i]*b[i];
}
```
-  d)

```
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;
```

5

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

Elección única

```
const int N = 5000, REP = 40000;
```

```
int R[REP + 1];
```

```
struct S { int a, b; } s[N];
```

Usuario Profesores

a) `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 ) ;
}

```



```

b) 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 );
}

```

•



```

c) 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);

```



```

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;
}

```

6

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

Elección única

Usuario Profesores

a) `a = b * c;`b) `a = c * b;`c) `a = b + b + b + b + b;`

•


d) `a = b + (b << 2);`


7


¿Cuál de los siguientes códigos es computacionalmente más eficiente?


Elección única

Usuario Profesores

-  a)

```
x = w & 7;
y = x * x;
z = (y << 5 )+y;
for (i = h = 0 ; i < MAX ; i++) {
    h += 14;
}
```
-  b)

```
x = w & 7;
y = pow (x, 2,0);
z = (y << 5 )+y;
for (i = h = 0 ; i < MAX ; i++) {
    h += 14;
}
```
-  c)

```
x = w % 8;
y = pow(x, 2.0);
z = y * 33;
for (i = 0 ; i < MAX ; i++) {
    h = 14 * i;
}
```
-  d)

```
x = w % 8;
y = x * x;
z = (y << 5 )+y;
for (i = 0 ; i < MAX ; i++) {
    h = 14 * i;
}
```


8


¿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.


Elección única


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

Usuario Profesores

-  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)
            #pragma omp atomic
            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)
            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];
```




9

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

Elección única

Usuario Profesores

-  a) Multiplicación, división y desplazamiento de bits





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

10

Elección única

¿Qué forma de ordenar las opciones *case* que aparecen en una sentencia *switch* ofrece mejores prestaciones?

Usuario Profesores

-  a) deben aparecer primero las opciones que contengan un mayor número de instrucciones
-  b) deben aparecer primero las opciones que contengan un menor número de instrucciones
-  c) el orden de aparición es indiferente
-  d) deben aparecer primero las opciones que tengan una mayor probabilidad de ser ciertas