

COMPILADORES E INTÉRPRETES

Generación y optimización de código

Práctica 2:

Lee todas las optimizaciones indicadas en el enunciado. Considera la que tienes asignada. Para ella mide los tiempos de ejecución de las versiones con y sin la optimización. Haz varias medidas de cada caso. Compila con la optimización -O0 para que el resultado no se aplican otras modificaciones de código que afecten al resultado. Haz un estudio para determinar la influencia del tamaño del problema N y, de haberlos, de otros parámetros del código que afecten al rendimiento. Para obtener tiempos de ejecución significativos, los lazos deben repetirse ITER veces como muestran los códigos (el valor de ITER debe ser suficiente para que el tiempo de ejecución sea del orden de una decena de segundos). ITER debe ser una constante del código.

Envía un breve informe al profesor para la técnica de optimización que hayas analizado, explicando en qué consiste, cual es el beneficio de la optimización e interpretando los resultados. Analizar el código ensamblador generado sirve para asegurar que se ha aplicado la técnica de optimización y puede ayudar a la interpretación de los resultados. El informe debe ser **ANÓNIMO** (no cumplir este requisito implica bajar la calificación en 1 punto).

En los códigos, el tipo de letra “for” se corresponde con la versión inicial del código, y el tipo “**for**” al de la mejora. Seguramente sea conveniente escribir ambas versiones en el mismo programa. Inicializa los valores de los datos utilizados. Intenta evitar en las medidas el efecto de las cargas iniciales de los datos en las caches.

1. Considera la optimización de **desenrolle de lazos internos**. Por ejemplo con el siguiente código para una profundidad d=4. Analiza la influencia del valor de d en los resultados:

```
int i, k;
float a=1.3, b=1.12;
float x[N], y[N];
for(k=0; k<ITER; k++){
    for(i=0; i<N; i++){
        y[i] = a * x[i] + b;
    }

for(k=0; k<ITER; k++){
    for(i=0; i<N; i+=4){
        y[i]   = a * x[i]   + b;
        y[i+1] = a * x[i+1] + b;
        y[i+2] = a * x[i+2] + b;
        y[i+3] = a * x[i+3] + b;
    }
}
```

2. Considera la optimización de búsqueda de **subexpresiones locales comunes**. Por ejemplo con el siguiente código:

```
int i, j;
float a, b, c, d, k, x, y, z, t;
float A[N];
for(j=0; j<ITER; j++)
  for(i=0; i<N; i++){
    a = x * y * z * t * A[i];
    b = x * z * t;
    c = y * t * A[i];
    d = k * A[i] - x*z;
    A[i] = (z * x * A[i]) + (y*t) + A[i];
  }
```

```
float tmp1, tmp2, tmp3, tmp4;
for(j=0; j<ITER; j++)
  for(i=0; i<N; i++){
    tmp1 = x * z;
    tmp2 = A[i];
    tmp3 = y * t;
    tmp4 = tmp1 * tmp2;
    a = tmp3 * tmp4;
    b = tmp1 * t;
    c = tmp3 * tmp2;
    d = k * tmp2 - tmp1;
    A[i] = tmp4 + tmp3 + tmp2;
  }
```

3. Considera la optimización de **resolver condicionales**. Por ejemplo con el siguiente código:

```
int i, k;
float a=1.3, x[N], y[N];
for(k=0; k<ITER; k++)
  for(i=0; i<N; i++)
    if (a*k==0.0) y[i] = 0;
    else y[i] = x[i]*y[i];

for(k=0; k<ITER; k++) {
  if(a*k==0.0) {
    for(i=0; i<N; i++)
      y[i] = 0;
    }
  else {
    for(i=0; i<N; i++)
      y[i] = x[i]*y[i];
    }
}
```

4. Considera la optimización de **paso de bucles dentro de una función**. Por ejemplo con el siguiente código:

```
int i, j;
float x[N], y[N], z[N];

void suma(float x, float y, float *z)
{
    *z = x + y;
}

void suma2(float *x, float *y, float *z)
{
int i, j;
for(j=0; j<ITER; j++)
    for(i=0; i<N; i++)
        z[i] = x[i] + y[i];
}

for(j=0; j<ITER; j++)
    for(i=0; i<N; i++){
        suma(x[i], y[i], &z[i]);
    }

suma2(x, y, z);
```

5. Considera la optimización de **fusión de lazos y aprovechamiento de localidad**. Por ejemplo con el siguiente código:

```
int i, j;
typedef struct {float x, y, z;} nuevotipo;
nuevotipo v1[N], v2[N], v3[N];
for(j=0; j<ITER; j++){
    for(i=0; i<N; i++)
        v3[i].x = v1[i].x + v2[i].x;
    for(i=0; i<N; i++)
        v3[i].y = v1[i].y - v2[i].y;
    for(i=0; i<N; i++)
        v3[i].z = v1[i].z * v2[i].z;
}

for(j=0; j<ITER; j++){
    for(i=0; i<N; i++) {
        v3[i].x = v1[i].x + v2[i].x;
        v3[i].y = v1[i].y - v2[i].y;
        v3[i].z = v1[i].z * v2[i].z;
    }
}
```

6. Considera la optimización de **fusión de lazos**. Por ejemplo con el siguiente código:

```
int i, j;
float x[N], y[N], z[N], k[N], t[N], u[N];
for(j=0; j<ITER; j++){
    for(i=0; i<N; i++){
        z[i] = x[i] + y[i];
        k[i] = x[i] - y[i];
        t[i] = x[i] * y[i];
        u[i] = z[i] + k[i] + t[i];
    }
}
```

```
float tmpx, tmpy;
for(j=0; j<ITER; j++){
    for(i=0; i<N; i++){ {
        tmpx = x[i];
        tmpy = y[i];
        z[i] = tmpx + tmpy;
        k[i] = tmpx - tmpy;
        t[i] = tmpx * tmpy;
        u[i] = z[i] + k[i] + t[i];
    }
}
```

7. Considera la optimización de **desenrolle de lazos internos y optimización de operaciones de reducción**. Por ejemplo con el siguiente código para una profundidad d=4. Analiza la influencia del valor de d en los resultados:

```
int i, k;
float x[N], y[N];
float a=0.0;
for(k=0; k<ITER; k++){
    a=0.0;
    for(i=0; i<N; i++){
        a = a + x[i] * y[i];
    }
}
```

```
for(k=0; k<ITER; k++){
    a = a1 = a2 = a3 = 0.0;
    for(i=0; i<N; i+=4){
        a = a + x[i] * y[i];
        a1 = a1 + x[i+1] * y[i+1];
        a2 = a2 + x[i+2] * y[i+2];
        a3 = a3 + x[i+3] * y[i+3];
    }
    a = a + a1 + a2 + a3;
}
```

8. Considera la optimización de **reemplazo de multiplicaciones y divisiones enteras por operaciones de desplazamiento**. Por ejemplo con el siguiente código:

```
int i, j, a, b=0;
for(j=0; j<ITER; j++)
  for(i=0; i<N; i++) {
    a = i * 8;
    b += a / 32;
  }

for(j=0; j<ITER; j++) {
  for(i=0; i<N; i++) {
    b += i >> 2;
  }
  a= (N-1) << 3; }
```

9. Considera la optimización de **peeling de lazos**. Por ejemplo con el siguiente código:

```
int i, k;
float x[N], y[N];
for(k=0; k<ITER; k++)
  for(i=0; i<N; i++)
    if(i==N/2) x[i] = 0;
    else if(i==N-1) x[i] = N-1;
    else x[i] = x[i]+y[i];

for(k=0; k<ITER; k++) {
  for(i=0; i<N/2; i++)
    x[i] = x[i]+y[i];
  x[N/2]=0;
  for(i=N/2+1; i<N-1; i++)
    x[i] = x[i]+y[i];
  x[N-1] = N-1;
}
```

10. Considera la optimización de **minimizar la sobrecarga de los condicionales**. Por ejemplo con el siguiente código:

```
int i, j, k;
float x[2*N][2*N], z[2*N][2*N];
for(k=0; k<ITER; k++)
  for(j=-N; j<N; j++)
    for(i=-N; i<N; i++){
      if(i*i+j*j != 0)
        z[N+j][N+i] = x[N+j][N+i] / (i*i+j*j);
      else
        z[N+j][N+i] = 0.0;
    }

for(k=0; k<ITER; k++) {
  for(j=-N; j<N; j++) {
    for(i=-N; i<0; i++)
      z[N+j][N+i] = x[N+j][N+i] / (i*i+j*j);
    for(i=1; i<N; i++)
      z[N+j][N+i] = x[N+j][N+i] / (i*i+j*j);
  }
  z[N][N] = 0.0; }
```

11. Considera la optimización de **desenrolle de lazos internos y optimización de reducciones**. Por ejemplo con el siguiente código para una profundidad d=4. Analiza la influencia del valor de d en los resultados:

```
int i, k;
float x[N];
float a;
for(k=0; k<ITER; k++){
    a=1.0;
    for(i=0; i<N; i++){
        a = a * x[i]; }

float a0, a1, a2, a3
for(k=0; k<ITER; k++){
    a = 1.0;
    for(i=0; i<N; i+=4){
        a0 = x[i];
        a1 = a0 * x[i+1];
        a2 = x[i+2];
        a3 = a2 * x[i+3];
        a = a * a1 * a3;
    }
}
```

12. Considera la optimización de **inlining**: sustituir la llamada a una función por el propio cuerpo de la función. Por ejemplo con el siguiente código:

```
int i, j;
float x[N], y[N], z[N];
void add(float x, float y, float *z)
{
    *z = x + y;
}
for(j=0; j<ITER; j++){
    for(i=0; i<N; i++){
        add(x[i], y[i], &z[i]);
    }

for(j=0; j<ITER; j++)
    for(i=0; i<N; i++)
        z[i] = x[i] + y[i];
```

13. Considera la optimización de **fusión de lazos**. Por ejemplo con el siguiente código:

```
int i, j, k;
float x[N], y[N];
for(k=0; k<ITER; k++){
    for(i=0; i<N; i++){
        x[i] = sqrt((float)i);
    }
    for(i=0; i<N; i++){
        y[i] = (float)i+2.0;
    }
    for(i=0; i<N; i++){
        x[i] += sqrt(y[i]);
    }
}

for(k=0; k<ITER; k++)
    for(i=0; i<N; i++) {
        x[i] = sqrt((float)i);
        y[i] = (float)i+2.0;
        x[i] += sqrt(y[i]);
    }
```

14. Considera la optimización de **reemplazo de divisiones en punto flotante por una sola división y multiplicaciones**. Por ejemplo con el siguiente código:

```
int i, j;
float a, b, c, d, t, u, v, x, y, z, tmp;
for(j=0; j<ITER; j++)
  for(i=0; i<N; i++){
    x = i+1.1;
    y = j+x;
    a = 1 / x;
    b = 1 / (x*y);
    c = 1 / y;
    d = v / (x*y);
  }

for(j=0; j<ITER; j++)
  for(i=0; i<N; i++){
    x = i+1.1;
    y = j+x;
    b = 1/(x*y);
    a = y * b;
    c = x * b;
    d = v * b;
  }
```

15. Considera la optimización de **unroll and jam**. Por ejemplo con el siguiente código para una profundidad d=4. Analiza la influencia del valor de d en los resultados:

```
int i, j, k, l;
float a[N][N], b[N][N], c[N][N];
for(l=0; l<ITER; l++)
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      for(k=0; k<N; k++)
        c[k][i] += a[k][j] * b[j][i];

for(l=0; l<ITER; l++)
  for(i=0; i<N; i+=2)
    for(j=0; j<N; j+=2)
      for(k=0; k<N; k++){
        c[k][i] += (a[k][j] * b[j][i] + a[k][j+1] * b[j+1][i]);
        c[k][i+1] += (a[k][j] * b[j][i+1] + a[k][j+1] * b[j+1][i+1]);
      }
```

16. Considera la optimización de **intercambio de lazos**. Por ejemplo con el siguiente código:

```
int i, j, k;
float x[N][N], y[N][N];
for(k=0; k<ITER; k++){
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      y[j][i] = x[j][i] * 3.0;
}

for(k=0; k<ITER; k++){
  for(j=0; j<N; j++)
    for(i=0; i<N; i++)
      y[j][i] = x[j][i] * 3.0;
  }
```