AULAS TEÓRICO-PRÁTICAS DE COMPILADORES

2º semestre de 2002/2003

Optimizações de código

AULA Nº 1 (2ª parte da aula: 1 hora e 30 minutos)

Esta aula tem como objectivo a utilização de algumas das optimizações de código realizadas pelos compiladores. Algumas dessas optimizações não dependem do processador alvo para garantirem sempre bons resultados. As optimizações do tipo anterior e cujo impacto será avaliado são:

- Propagação de constantes (*Constant propagation*);
- Avaliação de expressões com constantes (*Constant folding* ou *Constant-Expression evaluation*);
- Simplificações algébricas (*Algebraic Simplifications*)
- Substituição de escalares (*Scalar replacement*)
- Redução do custo (*Strength reduction*)
- Eliminação de sub-expressões comuns (*Common-Subexpression elimination* - CSE)
- Eliminação de código e de declarações não utilizadas
- Desenrolamento de ciclos (*Loop unrolling*)

Após leitura desta ficha e verificação das melhorias no tempo de execução, descreva cada uma das optimizações anteriores.

1 Verificar a diferença entre o tempo de execução utilizando o interpretador e utilizando o compilador JIT

O programa exemplo que vamos utilizar é formado pelas classes em bytecodes e por uma classe para a qual é dada o ficheiro Java original. Para executar o programa deve primeiro compilar o ficheiro Filter.java e depois executar um dos comandos seguintes:

Utilizando o interpretador:

```
Java –Xint DrawImage
```

Utilizando o compilador JIT (por omissão):

```
Java DrawImage
```

Verifique os tempos de execução de cada um dos algoritmos apresentados.

2 Optimizações

Vamos considerar a função `doFIR` apresentada em baixo e que é um método da classe `Filter.java`. Este método, como pôde constatar anteriormente implementa um algoritmo de processamento de imagem que reduz o ruído de uma imagem. De seguida vamos realizar manualmente um conjunto

de optimizações, que integram ou constam como opções de compiladores e que permitem melhorar o desempenho.

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    int DIM = 350;
    short[] OUT = new short[DIM*DIM];

    for (int row=0; row < DIM-3+1; row++) {
        for (int col = 0; col< DIM -3+1; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol<3; wcol++) {
                    sumval  += IN[(row +wrow)*DIM +(col+wcol)]*K[wrow*3+wcol];
                }
            }
            sumval = sumval / 16;
            OUT[row * DIM  + col] =  (short) sumval;
       }
    }
    return OUT;
}
```

2.1 Propagação de constantes (Constant Propagation)

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    short[] OUT = new short[350*350];

    for (int row=0; row < 350-3+1; row++) {
        for (int col = 0; col< 350-3+1; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol<3; wcol++) {
                    sumval  += IN[(row +wrow)*350 +(col+wcol)]*K[wrow*3+wcol];
                }
            }
            sumval = sumval / 16;
            OUT[row * 350  + col] =  (short) sumval;
       }
    }
    return OUT;
}
```

2.2 Constant folding (Constant-Expression Evaluation)

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    short[] OUT = new short[350*350];   ————————   12250

    for (int row= 0; row < 350-3+1; row++) {   ——   348
        for (int col = 0; col< 350-3+1; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol<3; wcol++) {
                    sumval+= IN[(row +wrow)*350+(col+wcol)]*K[wrow*3+wcol];
```

```
                }
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] =  (short) sumval;
        }
    }
    return OUT;
}
```

Depois de aplicar *constant folding*:

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    short[] OUT = new short[12250];

    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol<3; wcol++) {
                    sumval+= IN[(row +wrow)*350+(col+wcol)]*K[wrow*3+wcol];
                }
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] =  (short) sumval;
        }
    }
    return OUT;
}
```

2.3 Loop Unrolling do ciclo

```
for (int wcol = 0; wcol<3; wcol++) {
    sumval+= IN[(row +wrow)*350+(col+wcol)]*K[wrow*3+wcol];
}
```

Obtém-se:

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    short[] OUT = new short[12250];

    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                sumval+= IN[(row +wrow)*350+(col+0)]*K[wrow*3+0];
                sumval+= IN[(row +wrow)*350+(col+1)]*K[wrow*3+1];
                sumval+= IN[(row +wrow)*350+(col+2)]*K[wrow*3+2];
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] =  (short) sumval;
        }
    }
    return OUT;
}
```

2.4 Simplificação algébrica (Algebraic simplification)

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    short[] OUT = new short[12250];

    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                sumval+= IN[(row +wrow)*350+col])*K[wrow*3];
                sumval+= IN[(row +wrow)*350+(col+1)])*K[wrow*3+1];
                sumval+= IN[(row +wrow)*350+(col+2)])*K[wrow*3+2];
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] =  (short) sumval;
       }
    }
    return OUT;
}
```

2.5 Loop Unrolling do ciclo

```
for (int wrow=0; wrow < 3; wrow++) {
    sumval+= IN[(row +wrow)*350+col]*K[wrow*3];
    sumval+= IN[(row +wrow)*350+(col+1)]*K[wrow*3+1];
    sumval+= IN[(row +wrow)*350+(col+2)]*K[wrow*3+2];
}
```

Obtém-se:

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    short[] OUT = new short[12250];

    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = 0;
            sumval+= IN[(row +0)*350+col]*K[0*3];
            sumval+= IN[(row +0)*350+(col+1)]*K[0*3+1];
            sumval+= IN[(row +0)*350+(col+2)]*K[0*3+2];
            sumval+= IN[(row +1)*350+col]*K[1*3];
            sumval+= IN[(row +1)*350+(col+1)]*K[1*3+1];
            sumval+= IN[(row +1)*350+(col+2)]*K[1*3+2];
            sumval+= IN[(row +2)*350+col]*K[2*3];
            sumval+= IN[(row +2)*350+(col+1)]*K[2*3+1];
            sumval+= IN[(row +2)*350+(col+2)]*K[2*3+2];
            sumval = sumval / 16;
            OUT[row * 350 + col] =  (short) sumval;
       }
    }
    return OUT;
}
```

2.6 Simplificações algébricas + constant folding

```
final public static short[] doFIR(short[] IN) {

    short[] K =  {1, 2, 1,
                  2, 4, 2,
                  1, 2, 1};

    short[] OUT = new short[12250];
```

```
        for (int row=0; row < 348; row++) {
            for (int col = 0; col< 348 ; col++) {
                int sumval= IN[row*350+col]*K[0];
                sumval+=   IN[row*350+(col+1)]*K[1];
                sumval+=   IN[row*350+(col+2)]*K[2];
                sumval+=   IN[(row +1)*350+col]*K[3];
                sumval+=   IN[(row +1)*350+(col+1)]*K[4];
                sumval+=   IN[(row +1)*350+(col+2)]*K[5];
                sumval+=   IN[(row +2)*350+col]*K[6];
                sumval+=   IN[(row +2)*350+(col+1)]*K[7];
                sumval+=   IN[(row +2)*350+(col+2)]*K[8];
                sumval = sumval / 16;
                OUT[row * 350 + col] =  (short) sumval;
            }
        }
        return OUT;
    }
```

2.7 Scalar replacement

```
    final public static short[] doFIR(short[] IN) {

        short[] K =  {1, 2, 1,
                      2, 4, 2,
                      1, 2, 1};

        short[] OUT = new short[12250];

        for (int row=0; row < 348; row++) {
            for (int col = 0; col< 348 ; col++) {
                int sumval= IN[row*350+col]*1;
                sumval+=   IN[row*350+(col+1)]*2;
                sumval+=   IN[row*350+(col+2)]*1;
                sumval+=   IN[(row +1)*350+col]*2;
                sumval+=   IN[(row +1)*350+(col+1)]*4;
                sumval+=   IN[(row+1)*350+(col+2)]*2;
                sumval+=   IN[(row +2)*350+col]*1;
                sumval+=   IN[(row +2)*350+(col+1)]*2;
                sumval+=   IN[(row +2)*350+(col+2)]*1;
                sumval = sumval / 16;
                OUT[row * 350 + col] =  (short) sumval;
            }
        }
        return OUT;
}
```

2.8 Simplificação algébrica

```
    final public static short[] doFIR(short[] IN) {

        short[] K =  {1, 2, 1,
                      2, 4, 2,
                      1, 2, 1};

        short[] OUT = new short[12250];

        for (int row=0; row < 348; row++) {
            for (int col = 0; col< 348 ; col++) {
                int sumval= IN[row*350+col];
                sumval+=   IN[row*350+col+1]*2;
                sumval+=   IN[row*350+col+2];
                sumval+=   IN[(row +1)*350+col]*2;
                sumval+=   IN[(row +1)*350+col+1]*4;
                sumval+=   IN[(row +1)*350+col+2]*2;
                sumval+=   IN[(row +2)*350+col];
                sumval+=   IN[(row +2)*350+col+1]*2;
                sumval+=   IN[(row +2)*350+col+2];
                sumval = sumval / 16;
                OUT[row * 350 + col] =  (short) sumval;
            }
```

```
        }
        return OUT;
}
```

2.9 Eliminação de código ou de declarações não utilizadas

```
    final public static short[] doFIR(short[] IN) {
        short[] OUT = new short[12250];

        for (int row=0; row < 348; row++) {
            for (int col = 0; col< 348; col++) {
                int sumval= IN[row*350+col];
                sumval+=  IN[row*350+col+1]*2;
                sumval+=  IN[row*350+col+2];
                sumval+=  IN[(row +1)*350+col]*2;
                sumval+=  IN[(row +1)*350+col+1]*4;
                sumval+=  IN[(row +1)*350+col+2]*2;
                sumval+=  IN[(row +2)*350+col];
                sumval+=  IN[(row +2)*350+col+1]*2;
                sumval+=  IN[(row +2)*350+col+2];
                sumval = sumval / 16;
                OUT[row * 350 + col] =  (short) sumval;
           }
        }
        return OUT;
    }
```

2.10 Strength reduction

```
    final public static short[] doFIR(short[] IN) {
        short[] OUT = new short[12250];

        for (int row=0; row < 348; row++) {
            for (int col = 0; col< 348; col++) {
                int sumval= IN[row*350+col];
                sumval+=  IN[row*350+col+1]<<1;
                sumval+=  IN[row*350+col+2];
                sumval+=  IN[(row +1)*350+col]<<1;
                sumval+=  IN[(row +1)*350+col+1]<<2;
                sumval+=  IN[(row +1)*350+col+2]<<1;
                sumval+=  IN[(row +2)*350+col];
                sumval+=  IN[(row +2)*350+col+1]<<1;
                sumval+=  IN[(row +2)*350+col+2];
                sumval = sumval >> 4;
                OUT[row * 350 + col] =  (short) sumval;
           }
        }
        return OUT;
    }
```

2.11 Depois de simplificações algébricas e reassociação

```
    final public static short[] doFIR(short[] IN) {
        short[] OUT = new short[12250];

        for (int row=0; row < 348; row++) {
            for (int col = 0; col< 348; col++) {
                int sumval=IN[row*350+col];
                sumval+=IN[row*350+col+1]<<1;
                sumval+=IN[row*350+col+2];
                sumval+=IN[350*row +350+col]<<1;
                sumval+=IN[350*row +351+col]<<2;
                sumval+=IN[350*row +352+col]<<1;
                sumval+=IN[350*row +700+col];
                sumval+=IN[350*row +701+col]<<1;
                sumval+=IN[350*row +702+col];
                sumval = sumval >> 4;
                OUT[row * 350 + col] =  (short) sumval;
           }
```

```
        }
        return OUT;
    }
```

2.12 Depois de CSE (eliminação de sub-expressões comuns):

```
final public static short[] doFIR(short[] IN) {
    short[] OUT = new short[12250];

    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {

            int row_350_col = row*350 + col;

            int sumval= IN[row_350_col];
            sumval+=  IN[row_350_col + 1]<<1;
            sumval+=  IN[row_350_col + 2];
            sumval+=  IN[row_350_col + 350]<<1;
            sumval+=  IN[row_350_col + 351]<<2;
            sumval+=  IN[row_350_col + 352]<<1;
            sumval+=  IN[row_350_col + 700];
            sumval+=  IN[row_350_col + 701]<<1;
            sumval+=  IN[row_350_col + 702];
            sumval = sumval >> 4;
            OUT[row_350_col] =  (short) sumval;
        }
    }
    return OUT;
}
```

3 Sumário

A utilização das optimizações consideradas nesta ficha permitem, sempre que haja potencial para a sua aplicação, melhorar o desempenho do código. No exemplo indicado foi necessário aplicar algumas das optimizações mais do que uma vez, em virtude da aplicação de certas optimizações potenciar a aplicação de outras (ver por exemplo os casos dos desenrolamentos dos dois ciclos).