Code Transformations and Optimizations

Compilers course

Masters in Informatics and Computing Engineering (MIEIC), 3rd Year

João M. P. Cardoso

Dep. de Engenharia Informática
Faculdade de Engenharia (FEUP), Universidade do Porto, Porto
Portugal
Email:jmpc@acm.org

Compilers

OTHER OPTIMIZATIONS

by João M. P. Cardoso

Optimizations

- There are many optimizations
  - Constant Propagation
  - *Strength Reduction*
  - Constant Folding
  - Common Subexpression Elimination
  - Scalar Replacement
  - Elimination of Redundant Memory Accesses: data reuse
  - Dead-Code Elimination
  - Code Movement
  - Data reuse
  - Loop transformations
    - *Loop Unrolling* and…
    - many others

Optimizations

Example (image smooth operation):

```
final public static void doFIR(short[] IN, short[] OUT) {
    int DIM = 350;
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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;
      }
    }
}
```

Optimizations

Constant Propagation

```
final public static void doFIR(short[] IN, short[] OUT) {
    int DIM = 350;
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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;
      }
    }
}
```

Optimizations

Constant folding (Constant-Expression Evaluation):

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 350-3+1; row++) {     348
      for (int col = 0; col< 350-3+1; col++) {     348
        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;
      }
    }
}
```

Optimizations

Loop Unrolling:

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

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

```
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];
```

Optimizations

Loop Unrolling:

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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;
      }
    }
}
```

Optimizations

Algebraic simplification:

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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;
      }
    }
}
```

Optimizações

Loop Unrolling:

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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;
      }
    }
}
```

Optimizations

Loop Unrolling:

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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; }}}
```

Optimizations

Algebraic simplifications+ *constant folding*:

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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;
}}}
```

Optimizations

Scalar Replacement:

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
     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;
     }}}
```

Optimizations

Code Elimination of declarations and initializations not used:

```
final public static void doFIR(short[] IN, short[] OUT) {
    short[] K =  {1, 2, 1, 2, 4, 2, 1, 2, 1};
    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;
    }}}
```

Optimizations

Code Elimination of declarations and initializations not used:

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

  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;
  }}}
```

Optimizations

Strength reduction:

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

    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;
    }}}
```

Optimizations

After algebraic optimizations and reassociation:

```
final public static void doFIR(short[] IN, short[] OUT) {
    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;
     }}}
```

Optimizations

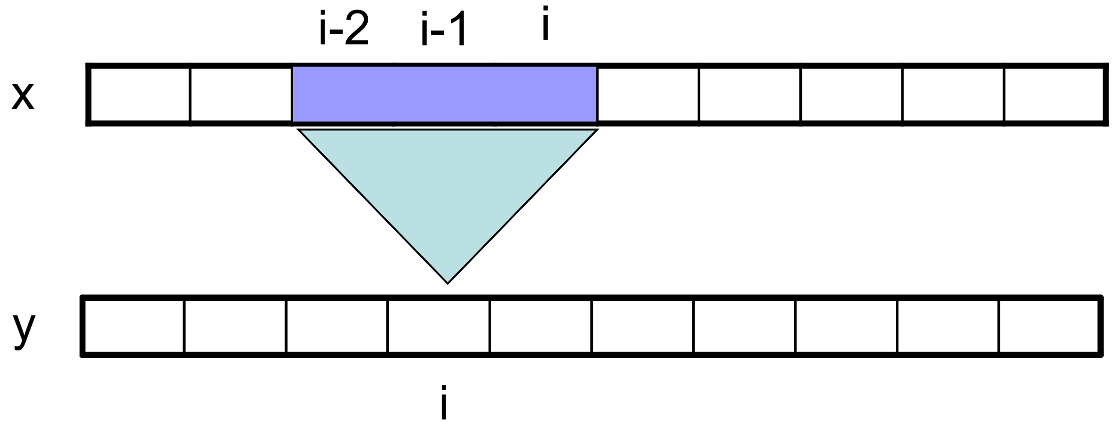
Common subexpression elimination:

```
final public static void doFIR(short[] IN, short[] OUT) {
    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;
    }}}
```

Optimizations

Data reuse:

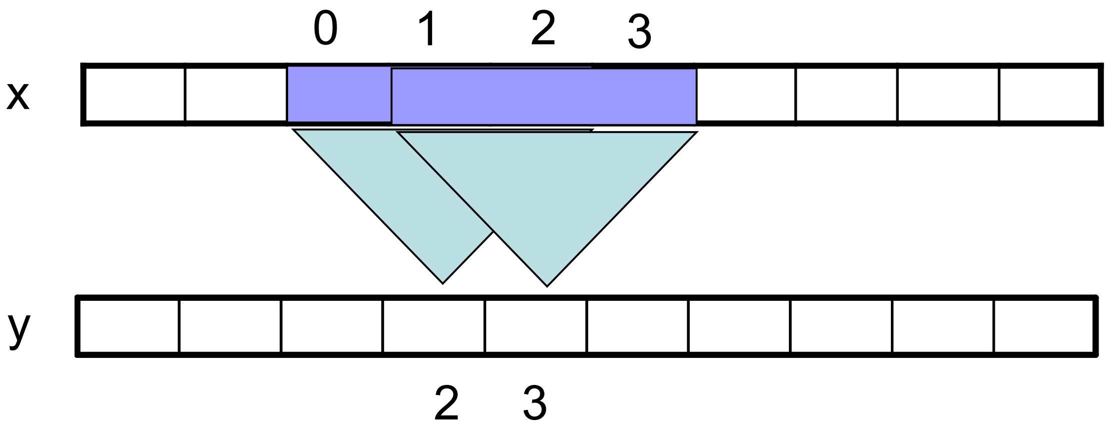
```
for (int i = 2; i< N; i++) {
   y[i] = x[i] + x[i-1] + x[i-2];
}
```



Optimizations

Data reuse:

```
for (int i = 2; i< N; i++) {
   y[i] = x[i] + x[i-1] + x[i-2];
}
```

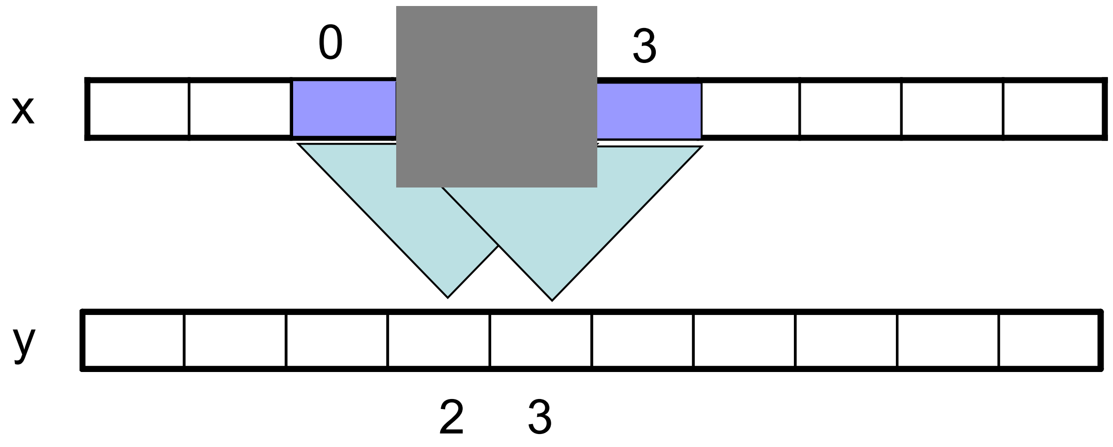


Optimizations

Data reuse:

```
for (int i = 2; i< N; i++) {
   y[i] = x[i] + x[i-1] + x[i-2];
}
```

```
int x_2 = x[0]
int x_1 = x[1];
Int x_0;
for (int i = 2; i< N; i++) {
   x_0 = x[i];
   y[i] = x_0 + x_1 + x_2;
   x_2 = x_1;
   x_1 = x_0;
}
```



Summary

- There are many optimizations and their impact depends on the code and on the target machine
- Usually, programmers use existent compiler flags that represent a set of optimizations (e.g., gcc’s -O2 and –O3)
- An important step is the selection of the optimizations to apply according to the code (e.g., of a function), target machine, and goal (e.g., performance, energy consumption)