Optimization exercise – multiplication of matrices

Matrix multiplication

A, B, C – matrices 20 x 20

C = A * B

c[i,j] = a[i,0]* b[0,j] + a[i,1]* b[1,j] + … + a[i,19]* b[19,j]

($0 \leq i, j \leq 19$)

Original program

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
   for (j = 0; j < 20; j++)
      for (k = 0; k < 20; k++)
           c[i,j] = c[i,j] + a[i,k] * b[k,j];
print(c)
```

Seems there is nothing to optimize…

But: a lot of optimizations can be done here!

Reveal address operations – done during intermediate code generation

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
  for (j = 0; j < 20; j++)
    for (k = 0; k < 20; k++)
        &(c + i*20*4 + j*4) =
                    ^(c + i*20*4 + j*4) + ^(a + i*20*4 + k*4) * ^(b + k*20*4 + j*4) ;
print(c)
```

NOTE: `^addr` – get value at address `addr`

Constant folding – perform operations on constant arguments

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
   for (j = 0; j < 20; j++)
      for (k = 0; k < 20; k++)
          &(c + i*20*4 + j*4) =
                    ^(c + i*20*4 + j*4) + ^(a + i*20*4 + k*4) * ^(b + k*20*4 + j*4) ;
print(c)
```

NOTE: 20*4 - performed 4*20*20*20 = 32,000 times !

Constant folding – result

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
  for (j = 0; j < 20; j++)
    for (k = 0; k < 20; k++)
        &(c + i*80 + j*4) =
                    ^(c + i*80 + j*4) + ^(a + i*80 + k*4) * ^(b + k*80+ j*4)
print(c)
```

Elimination of common sub-expressions

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
   for (j = 0; j < 20; j++)
      for (k = 0; k < 20; k++)
         &(c + i*80 + j*4) =
                     ^(c + i*80 + j*4) + ^(a + i*80 + k*4) * ^(b + k*80+ j*4)
print(c)
```

Elimination of common sub-expressions - result

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
  for (j = 0; j < 20; j++)
    for (k = 0; k < 20; k++)
       { t_i80 = i*80;
         t_j4 = j*4;
         c_i_j_addr = c + t_i80 + t_j4;
         &( c_i_j_addr) = ^( c_i_j_addr) + ^(a + t_i80 + k*4) * ^(b + k*80 + t_j4);
       }
print(c)
```

Loop-invariant code motion

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
  for (j = 0; j < 20; j++)
    for (k = 0; k < 20; k++)
       { t_i80 = i*80;
        t_j4 = j*4;
         c_i_j_addr = c + t_i80 + t_j4;
         &( c_i_j_addr) = ^( c_i_j_addr) + ^(a + t_i80 + k*4) * ^( b + k*80 + t_j4);
       }
print(c)
```

Loop-invariant code motion - result

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
  { t_i80 = i*80;
   a_i_row = a + t_i80;
   for (j = 0; j < 20; j++)
     {t_j4 = j*4;
      c_i_j_addr = c + t_i80 + t_j4;
      b_j_column = b + t_j4;
      for (k = 0; k < 20; k++)
        &( c_i_j_addr) = ^( c_i_j_addr) + ^(a_i_row + k*4) * ^( b_j_column + k*80);
     }
  }
print(c)
```

Operation strength reduction

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
for (i = 0; i < 20; i++)
  {t_i80 = i*80;              /*  values: 0, 80, 160, ... , 1520  */
  a_i_row = a + t_i80;
  for (j = 0; j < 20; j++)
    { t_j4 = j*4;
      c_i_j_addr = c + t_i80 + t_j4;
      b_j_column = b + t_j4;
      for (k = 0; k < 20; k++)
        &( c_i_j_addr) = ^( c_i_j_addr) + ^(a__i_row + k*4) * ^( b_j_column + k*80);
    }
  }                      /*  values: 0, 4, 8, ... , 76            values: 0, 80, 160, ... , 1520  */
print(c)
```

Operation strength reduction – result (use addition instead of multiplication)

```
int a[20,20], b[20,20], c[20,20] = {0};
read (a, b);
t_i80 = -80;
for (i = 0; i < 20; i++)
  {t_i80 = t_i80 + 80;                    /*  values: 0, 80, 160, ... , 1520   */
  a_i_row = a + t_i80;
  for (j = 0; j < 20; j++)
    { t_j4 = j*4;
      c_i_j_addr = c + t_i80 + t_j4;
      b_column = b + t_j4;
      t_k4 = -4;
      t_k80 = -80;
      for (k = 0; k < 20; k++)
        { t_k4 = t_k4 + 4;                /*  values: 0, 4, 8, ... , 76  */
         t_k80 = t_k80 + 80;              /*  values: 0, 80, 160, ... , 1520  */
        &( c_i_j_addr) = ^( c_i_j_addr) + ^(a_i_row + t_k4) * ^( b_j_column + t_k80);
    }
  }
print(c)
```