Optimising Compilers: Exercises 5

float a[N][M], b[N][M], c[N][M];

void h()
{
    int i;
    int j;

    for (i = 2; i < 100; i++) {
        for (j = 2 + 3 * i; j < 1000 - i; j++) {
            a[i][j] = a[i - 2][j + 3];
            b[i][j] = b[i][j - 2];
            c[i][j] = c[i + 1][j + 2];
        }
    }
}

Figure 1: Example loop.

1. Show the distance matrix D of the loop L above.
2. Find a unimodular matrix U so that the inner loop can execute in parallel.
3. Find the loop limits of the new loop L_U.

Solutions

1. There are three pairs of references, and we should do data dependence analysis for each pair. For each pair, let the reference on the left be denoted A and the one on the right be denoted B. We need to determine whether the equation

\[ IA + a_0 = JB + b_0 \]

has a solution.

For the matrix a we find the data dependence and the dependence distance as follows. Firstly, we have

\[ A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \]

and

\[ a_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \]

1
and

\[ B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \]

and

\[ b_0 = \begin{pmatrix} -2 & 3 \end{pmatrix}. \]

Here \( I = (i_1, j_1) \) and \( J = (i_2, j_2) \) represent the index variables. The equation becomes

\[ ( I; J ) \begin{pmatrix} A \\ -B \end{pmatrix} = b_0 - a_0. \]

or

\[ \begin{pmatrix} i_1 & j_1 & i_2 & j_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} -2 & 3 \end{pmatrix} - \begin{pmatrix} 0 & 0 \end{pmatrix}. \]

We see that

\[ i_1 - i_2 = -2 \]

and

\[ j_1 - j_2 = 3. \]

Let us write \( i_1 = t_1 \) and \( i_2 = t_1 + 2. \) Also, \( j_1 = t_2 \) and \( j_2 = t_2 - 3. \) Thus \( I = (t_1, t_2) \) and \( J = (t_1 + 2, t_2 - 3) \) is a solution to the dependence equation, which we can see lies within the loop bounds. The dependence distance, in general, \( d \) is \( 0 \) if \( I = J, \) \( I - J, \) if \( J < I, \) and \( J - I, \) if \( I < J. \) In our case \( d = (2, -3), \) which means the write instruction accesses a particular matrix element before that element is accessed by the read instruction.

For matrix \( b, \) we have

\[ A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \]

and

\[ a_0 = \begin{pmatrix} 0 & 0 \end{pmatrix}, \]

and

\[ B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \]

and

\[ b_0 = \begin{pmatrix} 0 & -2 \end{pmatrix}. \]
The equation becomes
\[
\begin{pmatrix}
I & J
\end{pmatrix}
\begin{pmatrix}
A \\
-B
\end{pmatrix} = b_0 - a_0.
\]
or
\[
\begin{pmatrix}
i_1 & j_1 & i_2 & j_2
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1 \\
-1 & 0 \\
0 & -1
\end{pmatrix} = \begin{pmatrix}
0 & -2 \\
0 & 0
\end{pmatrix}.
\]

We see that
\[i_1 - i_2 = 0\]
and
\[j_1 - j_2 = -2.\]
Let us write \(i_1 = t_1\) and \(i_2 = t_1\). Also, \(j_1 = t_2\) and \(j_2 = t_2 + 3\). Thus \(I = (t_1, t_2)\) and \(J = (t_1, t_2 + 2)\) is a solution to the dependence equation, which we again can see lies within the loop bounds. The dependence distance is \(d = (0, 2)\), which again means the write accesses a particular matrix element before that element is accessed by the read.

Finally, for matrix \(c\), we have
\[
A = \begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
and
\[
a_0 = \begin{pmatrix}
0 & 0
\end{pmatrix},
\]
and
\[
B = \begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}
\]
and
\[
b_0 = \begin{pmatrix}
1 & 2
\end{pmatrix}.
\]

The equation becomes
\[
\begin{pmatrix}
I & J
\end{pmatrix}
\begin{pmatrix}
A \\
-B
\end{pmatrix} = b_0 - a_0.
\]
or
\[
\begin{pmatrix}
i_1 & j_1 & i_2 & j_2
\end{pmatrix}
\begin{pmatrix}
1 & 0 \\
0 & 1 \\
-1 & 0 \\
0 & -1
\end{pmatrix} = \begin{pmatrix}
1 & 2 \\
0 & 0
\end{pmatrix}.
\]
We see that

\[ i_1 - i_2 = 1 \]

and

\[ j_1 - j_2 = 2. \]

Let us write \( i_1 = t_1 \) and \( i_2 = t_1 - 1 \). Also, \( j_1 = t_2 \) and \( j_2 = t_2 - 2 \). Thus \( I = (t_1, t_2) \) and \( J = (t_1 - 1, t_2 - 2) \) is a solution to the dependence equation, which we again can see lies within the loop bounds. To determine the dependence distance, we check whether \( J - I \) or \( I - J \) gives a lexicographically positive distance vector. In this case, \( 0 \prec I - J \), so, the dependence distance is \( d = (1, 2) \), and this time the read accesses a particular matrix element before that element is accessed by the write.

The distance matrix becomes

\[
D = \begin{pmatrix} 2 & -3 \\ 0 & 2 \\ 1 & 2 \end{pmatrix}.
\]

2. We want to find a unimodular matrix \( U \) so that for \( D_U = DU \) each element in the first column is at least one. Let \( u = (u_1, u_2, \ldots, u_m) \) denote the first column of \( U \) and let the rows of \( D \) be denoted by \( d_i \). Thus for each \( d_i \) we must have \( d_i u \geq 1 \).

With the distance matrix found in the previous question, no loop \( L_i \) can be executed in parallel. Searching for a transformation, we get the following system of inequalities:

\[
\begin{align*}
2u_1 & - 3u_2 \geq 1 \\
2u_2 & \geq 1 \\
u_1 & + 2u_2 \geq 1
\end{align*}
\]

We interchange the first two equations:

\[
\begin{align*}
2u_1 & - 3u_2 \geq 1 \\
u_1 & + 2u_2 \geq 1
\end{align*}
\]

Since there are no upper bounds on \( u_i \), there are infinitely many solutions to this equation. We choose the smallest integer \( u_i \) which satisfies the inequalities. So, \( u_2 \) is chosen as \([1/2] = 1\). Then we proceed with \( u_1 \), for which there are two inequalities: \( u_1 \geq [(1 + 3u_2)/2] = 2 \) and \( u_1 \geq [(1 - 2u_2)] = -1 \), so \( u_1 \) is chosen as the maximum of these, or \( u_1 = 2 \). We get

\[
U = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}
\]
and

\[ D \times U = \begin{pmatrix} 1 & 2 \\ 2 & 0 \\ 4 & 1 \end{pmatrix}. \]

The new loop nest \( L_U \) thus carries all dependences in the outermost loop \( L_1 \), with the consequence that \( L_2 \) can execute in parallel.

3. To find the loop limits of the transformed loop \( L_U \), we first express the original index variables (ie, \( i \) and \( j \)) using the new:

\[ I = KU^{-1} = (k_1 k_2) \begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix}. \]

This gives \( i = k_2 \) and \( j = k_1 - 2k_2 \). We then insert this into the loop bound inequalities of the original loop:

\[ 2 \leq i \leq 99 \]

and

\[ 2 + 3i \leq j \leq 999 - i. \]

That is,

\[ 2 \leq k_2 \leq 99 \]

and

\[ 2 + 3k_2 \leq k_1 - 2k_2 \leq 999 - k_2. \]

Solving this using Fourier-Motzkin elimination gives:

\[ 12 \leq k_1 \leq 1098 \]

and

\[ \max(2, k_1 - 999) \leq k_2 \leq \min(99, \lceil (-2 + k_1)/5 \rceil). \]

The resulting program is demonstrated in on the following page.
#include <stdio.h>

#define MIN(a, b) ((a)<(b)?(a):(b))
#define MAX(a, b) ((a)>(b)?(a):(b))

int main()
{
    int i, j;
    int k1, k2;
    int sum;
    int iterations;

    sum = iterations = 0;
    for (i = 2; i < 100; ++i) {
        for (j = 2 + 3 * i; j < 1000 - i; ++j) {
            sum += i * j;
            iterations += 1;
            //printf("S(%d, %d)\n", i, j);
        }
    }

    printf("Original loop:\n");
    printf("sum = %d\n", sum);
    printf("iterations = %d, iterations\n", sum);
    printf("\n");

    sum = iterations = 0;
    for (k1 = 12; k1 <= 1098; ++k1) {
        for (k2 = MAX(2, k1-999); k2 <= MIN(999, (-2+k1)/5); ++k2) {
            i = k2;
            j = k1 - 2 * k2;
            sum += i * j;
            iterations += 1;
            //printf("T(%d, %d)\n", i, j);
        }
    }

    printf("New loop:\n");
    printf("sum = %d\n", sum);
    printf("iterations = %d, iterations\n", iterations);

    return 0;
}