

CS 243 Assignment 6
Parallelization and Locality Optimizations
Sample Solutions and Grading Rubric

March 18, 2008

1. (30 points total)

a. (4 points)

The iteration space looks like a right triangle in the (i,j) plane with width and height equal to 1000 covering all points with $0 \leq i < 1000$, $0 \leq j < 1000$, and $i < j$. In order to get full credit, you must also draw the data dependences, which run along the diagonals with slope 1 (more precisely, the arrows should point up and to the right at 45 degree angles, but you don't need to draw it for full credit).

b. (3 points) - $[-1 \ 1] * [i \ j]^T + [0]$

c. (3 points) - Rank = 1

d. (3 points) - Nullity = 1

e. (3 points) - $[1 \ 1]^T$

f. (4 points)

$(0 \leq i < 1000) \wedge (0 \leq i' < 1000) \wedge (i \leq j < 1000) \wedge (i' \leq j' < 1000) \wedge$
 $(j - i = j' - i') \wedge ((j \neq j') \vee (i \neq i'))$

Note that the final clause states that either j does not equal j' OR i does not equal i'. Points were marked off for not acknowledging this point.

g. (3 points) - Yes

h. (4 points)

We accepted $p = j - i$ as well as $p = i - j$, which correspond to the following in matrix form, respectively:

$$p = [-1 \ 1] * [i \ j]^T + [0]$$
$$p = [1 \ -1] * [i \ j]^T + [0]$$

i. (3 points) - Anything mentioning the redundancies of doing multiple writes of the same constant value will get full credit.

2. (25 points total)

a. (3 points) - 1 degree of communication-free parallelism in the entire program. (Even though the second loop has 2 degrees by itself, there's a dependence between s1 in the first loop and s2 in the second loop due to the shared access of A[])

b. (7 points)

$p = i$, or in matrix form:

$$p = [1] * [i] + [0]$$

c. (7 points)

$p = k$, or in matrix form:

$$p = [0 \ 1] * [j \ k]^T + [0]$$

d. (8 points)

We gave full credit for both the un-optimized and optimized versions:

Un-optimized:

```
for (p=0; p<1000; p++) {
  for (i=0; i<1000; i++) {
    if (p == i) {
      A[i] = i;
    }
  }
  for (j=0; j<1000; j++) {
    for (k=0; k<1000; k++) {
      if (p == k) {
        B[j,k] = A[k] * B[j,k];
      }
    }
  }
}
```

Optimized:

```
for (p=0; p<1000; p++) {
  A[p] = p;
  for (j=0; j<1000; j++) {
    B[j,p] = A[p] * B[j,p];
  }
}
```

3. (20 points total)

a.) (6 points)

$10,000^2$ cache misses. Since the cache is too small to fit 10,000 entries and we are traversing one column at a time but the array is laid out in row-major order, every access will miss.

b.) (8 points)

2 possibilities:

- Permute the loop indices so that the inner loop (ranging over j) becomes the outer loop and vice versa
- Permute the array access indices to $A[i,j]$ instead of $A[j,i]$

c.) (6 points)

Since we are now traversing the array one row at a time, only one in four accesses will miss (assuming that integers are 1 word in size), so the number of misses will be:

$10,000^2 / 4$

4. (25 points total)

The first step to approach this problem is to identify the dependencies for each pair of accesses.

The accesses in this program are

1. Write access of $A[j,i]$ in statement 1.
2. Write access of $A[j,k]$ in statement 2.
3. Read access of $A[j,i]$ in statement 1.
4. Read access of $A[i,i]$ in statement 1.
5. Read access of $A[j,k]$ in statement 2.
6. Read access of $A[j,i]$ in statement 2.
7. Read access of $A[i,k]$ in statement 2.

The loop nest of statement 1 is 2 levels deep. Thus, the iteration space of the accesses of statement 1 is in the two dimensional space over integers, Z^2 . Similarly, the loop nest of statement 2 is 3 levels deep. Thus, the iteration space of the accesses of statement 2 is in the three dimensional space over integers, Z^3 .

The potential set of dependencies are all pairs of access a_1 and a_2 such that either one of a_1 or a_2 is a write access and that there exists a loop index vector i_1 in the iteration space of a_1 , and a loop index vector i_2 in the iteration space of a_2 , such that the location accessed by a_1 on iteration i_1 is the same as the location accessed by a_2 on iteration i_2 . Additionally, if a_1 and a_2 are in the same statement, then $i_1 \neq i_2$.

Consider loop 1.

The pair of accesses 1 and 2 are independent if and only if there does not exist (i,j) and (i',j',k') such that

1. $j=j'$
2. $i=k'$
3. $1 \leq i < 10000$
4. $i+1 \leq j < 10000$
5. $1 \leq i' < 10000$
6. $i'+1 \leq j' < 10000$
7. $i'+1 \leq k' < 10000$

Because these constraints are satisfiable, then the pair of accesses 1 and 2 are dependent.

The pair of accesses 1 and 7 are independent if and only if there does not exist (i,j) and (i',j',k') such that

1. $i' = j$
2. $k' = i$
3. $1 \leq i < 10000$
4. $i+1 \leq j < 10000$
5. $1 \leq i' < 10000$

6. $i'+1 \leq j' < 10000$
7. $i'+1 \leq k' < 10000$

Because these constraints are *not* satisfiable, then the pair of accesses 1 and 7 are independent.

The remaining dependencies can be found similarly by writing out and solving the corresponding integer linear programs.

Now, in order to determine whether loop 1 is parallelizable, we determine the affine partition constraints for the given the dependencies. Consider the dependency between accesses 2 and 7. Let $\langle C1, c1 \rangle$ be the affine partition of statement 1 and $\langle C2, c2 \rangle$ the affine partition of statement 2. Then, the affine partition constraint induced by the dependency between accesses 2 and 7 is:

$$C2 * i1 + c2 = C2 * i2 + c2$$

where $i1 = (i, j, k)$ and $i2 = (i', j', k')$. The $C2$ matrix must have 3 columns because the index vector has 3 rows. We initially assume $C2$ has 1 row regardless of the actual degrees of parallelization. The maximum number of linearly independent solutions to $C2$ will be the degrees of parallelization. Writing out the constraint in non-matrix form:

$$C21 * i + C22 * j + C23 * k + c2 = C21 * i' + C22 * j' + C23 * k' + c2.$$

The constraints on the loop index vectors $i1$ and $i2$ are $j' = i$ and $k' = k$. Applying this substitution yields:

$$C21 * i + C22 * j + C23 * k + c2 = C21 * i' + C22 * i + C23 * k + c2.$$

Then, writing the constraint yields:

$$(C21 - C22) * i + C22 * j - C21 * i' = 0$$

This requires that

$$C21 - C22 = 0$$

$$C22 = 0$$

$$-C21 = 0;$$

The solution that satisfies these constraints is that $C21 = C22 = 0$ and $C23$ is unconstrained. Consequently, the $C2$ has 1 linearly independent solution, namely $[0, 0, 1]$. Another solution to $C2$ is $[0, 0, 5]$ but this is not linearly independent from $[0, 0, 1]$. A single linearly independent solution means that this loop has at most one degree of parallelism.

In order to be certain that it has exactly 1 degree, we must consider each of the remaining dependencies. So, next consider the dependency between accesses 2 and 6.

Writing out the constraint in non-matrix form:

$$C21 * i + C22 * j + C23 * k + c2 = C21 * i' + C22 * j' + C23 * k' + c2.$$

The constraints on the loop index vectors i_1 and i_2 are $j' = j$ and $k' = i$. Applying this substitution yields:

$$C21 * i + C22 * j + C23 * k + c2 = C21 * i' + C22 * j + C23 * i + c2.$$

Then, writing the constraint yields:

$$(C21 - C23) * i + C23 * k - C21 * i' = 0$$

This requires that

$$C21 - C23 = 0$$

$$C23 = 0$$

$$-C21 = 0$$

in addition to the constraints from the dependency between 2 and 7:

$$C21 - C22 = 0$$

$$C22 = 0$$

$$-C21 = 0;$$

The only solution that satisfies these equations is that $C21 = C22 = C23 = 0$. Consequently, we are now sure that there are exactly zero degrees of parallelization in loop 1 because the only solution to C2 is $[0, 0, 0]$ which has rank 0. Considering any additional dependencies is not required since we're at the minimum possible degrees of parallelization.

Now, after concluding that loop 1 is not parallelizable, we consider loop 2 which allows us to fix i to some arbitrary constant between 1 and 10000, and completely ignore loop 1. Because of this relaxation, we must reconsider the dependencies between accesses. Observe that the iteration space of statement 1 is now 1-dimensional and the iteration space of statement 2 is now 2-dimensional.

The pair of accesses 1 and 2 are independent if and only if there does not exist (j) and (j',k') such that

1. $j=j'$
2. $i=k'$
3. $1 \leq i < 10000$
4. $i+1 \leq j < 10000$
5. $i+1 \leq j' < 10000$
6. $i+1 \leq k' < 10000$

Because these constraints are *not* satisfiable, then the pair of accesses 1 and 2 are independent.

Note how the loop index i of the outer loop is being treated as a constant in this set of constraints. The pair of accesses 2 and 6 are also independent by the same reasoning. The pair of accesses 1 and 7 are independent as shown earlier with a stronger set of constraints.

The pair of accesses 2 and 7 are independent if and only if there does not exist (j) and (j',k') such that

1. $j=i$
2. $k=k'$
3. $1 \leq i < 10000$
4. $i+1 \leq j < 10000$
5. $i+1 \leq j' < 10000$
6. $i+1 \leq k' < 10000$

Because these constraints are *not* satisfiable, then the pair of accesses 2 and 7 are independent.

The pair of accesses 1 and 6 are independent if and only if there does not exist (j) and (j',k') such that

1. $j=j'$
2. $1 \leq i < 10000$
3. $i+1 \leq j < 10000$
4. $i+1 \leq j' < 10000$
5. $i+1 \leq k' < 10000$

Because these constraints are satisfiable, then the pair of accesses 1 and 6 are dependent. In fact, this pair is the dependent pair of accesses.

Now, in order to determine whether loop 2 is parallelizable, we determine the affine partition constraints for the given the dependency between 1 and 6. Let $\langle C1, c1 \rangle$ be the affine partition of statement 1 and $\langle C2, c2 \rangle$ the affine partition of statement 2. Then, the affine partition constraint induced by the dependency between accesses 2 and 7 is:

$$C2 * i1 + c2 = C2 * i2 + c2$$

where $i1 = (j)$ and $i2 = (j',k')$. The $C1$ matrix must have 1 column because the index vector $i1$ has 1 row, and the $C2$ matrix must have 2 columns because the index vector $i2$ has 2 rows. We initially assume $C1$ and $C2$ have 1 row regardless of the actual degrees of parallelization. The maximum number of linearly independent solutions to $C1$ and $C2$ will be the degrees of parallelization. Writing out the constraint in non-matrix form:

$$C11 * j + c1 = C21 * j' + C22 * k' + c2.$$

The constraints on the loop index vectors $i1$ and $i2$ are $j' = j$. Applying this substitution yields:

$$C11 * j + c1 = C21 * j + C22 * k' + c2.$$

Then, writing the constraint yields:

$$(C11 - C21) * j - C22 * k' + c1 - c2 = 0$$

This requires that

$$C11 - C21 = 0$$

$$-C22 = 0$$

$$c1 - c2 = 0;$$

which means that $C11 = C21$, $c1 = c2$, and $C22 = 0$. The maximum number of linearly independent solutions of C2 is 1 because C22 is 0, so regardless of what C21 is, there will only be one linearly independent solution. Similarly, the maximum number of linearly independent solutions of C1 is 1. So, we can pick any solution to the equalities above: $C11 = 1$, $C21 = 1$, $c1 = 0$, $c2 = 0$. Consequently, the degrees of parallelization is 1, and thus the loop is parallelizable. Finally, $p = j$ for both statements 1 and 2.

The generated unoptimized SPMD code is:

```
for (i = 1; i < 10000; i++) { /* Loop1 */
    for (p = i + 1; p < 10000; p++) {
        for (j = i + 1; j < 10000; j++) { /* Loop2 */
            if (p == j) {
                A[j,i] = A[j,i] / A[i,i]; /* s1 */
            }

            for (k = i+1, k < 10000; k++) { /* Loop3 */
                if (p == j) {
                    A[j,k] = A[j,k] - A[j,i]*A[i,k]; /*s2 */
                }
            }
        }
    }
}
```