CSE 5441

Homework 2

Autumn 2014

Due 2:00PM, Friday, 9/19/2014 (submit via Carmen)

1. (30 points) Consider the following computation:

```
double A[N][N];
  for (i=0; i<N; i++)
   for (j=0; j<N; j++)
    s += A[j][i];
```

   Consider a processor with a cache with capacity of 8 Mbytes (i.e., 8 x 1024 x 1024 bytes) and linesize of 64 bytes. Assume N=2048 and that each element of A occupies 8 bytes. For each part below, perform cache miss analysis, assuming an empty cache before execution of the code:

   (a) A direct-mapped cache

   (b) A 4-way set-associative cache

   (c) A fully associative cache

   (d) What is a lower bound on the number of cache misses for this computation, as a function of N (i.e., what is the smallest possible number of misses irrespective of how large the cache is relative to $8N^2$ bytes)?

   (e) For N=2048, how large must a direct-mapped cache be in order that the number of cache misses equal the lower bound for the computation?

   (f) For N=2048, what is the minimum degree of associativity needed to minimize the number of cache misses (to the lower bound for the computation)?

2. (30 points) Consider the following code:

```
double A[N][N][N], C[N][N][N], B[N][N];

  for (i=0; i<N; i++)
   for (j=0; j<N; j++)
    for (k=0; k<N; k++)
     for (l=0; l<N; l++)
       C[k][j][i] += A[l][i][j]*B[k][l];
```

   Assume the loop nest is fully permutable, i.e., all permutations are allowed.

   (a) Perform stride analysis, considering each of the loops as the inner loop.

(b) For the above loop, perform independent cache miss analysis for each array, for a direct-mapped 2 Mbyte cache with linesize of 64 bytes, and N=128.

(c) Repeat the cache miss analysis for the lijk permutation of the loops.

3. (20 points) The following code performs matrix multiplication of an upper-triangular matrix A (it has zero values in its lower-triangular portion below the diagonal) with a full matrix B (i.e., not triangular) and the resulting matrix C is also full. To avoid unnecessary computations involving zero values, the k-loop only runs upwards from i.

```
for (i=0;i<1024;i++)
 for (j=0;j<1024;j++)
  for (k=i;k<1024;k++)
    C[i][j] += A[i][k]*B[k][j];
```

Generate the kij form for this computation (Hint: Think of the needed permutation as a sequence of "simpler" permutations for which the code generation is easier to reason about).

4. (20 points) For the same code as the previous problem, generate a version that is 2-way unrolled on both the i and j loops.

```
for (i=0;i<1024;i++)
 for (j=0;j<1024;j++)
  for (k=i;k<1024;k++)
    C[i][j] += A[i][k]*B[k][j];
```