IN3200/IN4200 Exercise Set 5

Exercise 1

Calculate code balance for the following loops, assuming that all arrays have to be loaded from memory. (Arrays `A`, `B` and `y` contain double-precision values.)

(a) Dense matrix-vector multiplication:

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

(b) Vector norm:

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

(c) Scalar product:

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

(d) Scalar product with indirect access:

```
double s = 0.;
for (i=0; i<N; i++)
  s += A[i]*B[K[i]];
```

Assume that each entry `K[i]` is a random integer between 0 and $N-1$.

Exercise 2

Please apply "unroll and jam" (with 4 as the unroll depth) to the outer loop of the following code:

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

Note: The value of `N` may not be divisible by 4.

Exercise 3

Please optmize the following function:

```
void foo (int N, double **mat, double **s, int *v) {
  int i,j;
  double val;

  for (j=0; j<N; j++)
    for (i=0; i<N; i++) {
      val = 1.0*(v[j]%256);
      mat[i][j] = s[i][j]*(sin(val)*sin(val)-cos(val)*cos(val));
    }
}
```

Assume that the above function will be repeatedly called, with changing contents of the input arrays `s` and `v`.

Exercise 4

Let us look at the sparse matrix-vector multiplication $C = C + A * B$, where the sparse matrix A is stored in the *compressed row storage* (CRS) format. That is, three arrays are used to store the nonzero values of A and remember where these nonzeros are located in the matrix:

- 1D array `val`, of length N_{nz}, stores all the nonzero values of the sparse matrix;

- 1D array `col_idx` , of length N_{nz}, records the original column positions of all the nonzero values;
- 1D array `row_ptr`, of length $N + 1$, contains the indices at which new rows start in array `val`. (N is the number of rows and columns in matrix A.)

The implementation of a sparse matrix-vector multiplication, using the CRS format for storing A, can thus be as follows:

```
for (i=0; i<N; i++) {
  tmp = C[i];
  for (j=row_ptr[i]; j<row_ptr[i+1]; j++)
    tmp = tmp + val[j]*B[col_idx[j]];
  C[i] = tmp;
}
```

It was shown in the slides for the third lecture of Chapter 3 that the "best-case" code balance for the above code segment is

$$\frac{N_{\text{nz}}(1 + 0.5) + 0.5N + N + 2N}{2N_{\text{nz}}} = \frac{3}{4} + \frac{7}{4}\frac{N}{N_{\text{nz}}}$$

What is the "worst-case" code balance?