(a) Naive Matrix-Vector Multiplication

```
void naive_mvm(float * A , float * x , float * y
      , int  N ) {
 for (int i = 0; i <  N ; ++i) {
   y [i] = 0;
  for (int j = 0; j <  N ; ++j) {
    y [i] +=  A [i *  N  + j] *  x [j];
  }
 }
}
```

(b) Unrolled Matrix-Vector Multiplication

```
void unrolled_mvm(float * A , float * x , float *
      y , int  N ) {
 for (int i = 0; i <  N ; ++i) {
   y [i] = 0;
  for (int j = 0; j <  N ; j += 4) {
    y [i] +=  A [i *  N  + j] *  x [j];
    y [i] +=  A [i *  N  + j + 1] *  x [j + 1];
    y [i] +=  A [i *  N  + j + 2] *  x [j + 2];
    y [i] +=  A [i *  N  + j + 3] *  x [j + 3];
  }
 }
}
```

(c) AVX Intrinsics Matrix-Vector Multiplication

```
#include <immintrin.h>

void avx_mvm(float * A , float * x , float * y
      , float  alpha , float  beta , int  lda
      , int  N ) {
 for (int i = 0; i <  N ; ++i) {
  __m256 vec_y = _mm256_setzero_ps();
  for (int j = 0; j <  N ; j += 8) {
   __m256 vec_a = _mm256_loadu_ps(& A [i *  lda
         + j]);
   __m256 vec_x = _mm256_loadu_ps(& x [j]);
   vec_y = _mm256_fmadd_ps(vec_a, vec_x, vec_y);
  }
   y [i] =  beta  *  y [i];
  float temp[8];
  _mm256_storeu_ps(temp, vec_y);
  for (int k = 0; k < 8; ++k) {
    y [i] +=  alpha  * temp[k];
  }
 }
}
```

(d) FPGA Accelerator Pass

```
void fpga_gemv(float * A , float * x , float * y
      , float  alpha , float  beta , int  lda
      , int  N ) {
 // Transfer data to FPGA
 transfer_to_fpga( A ,  x ,  y ,  alpha ,  beta
       ,  lda ,  N );

 // FPGA computation
 start_fpga_computation();

 // Retrieve results from FPGA
 retrieve_from_fpga( y );
}
```

Fig. 1. Matrix-Vector Multiplication Implementations and FPGA Accelerator Pass