/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void scale(float \* array, float factor, float offset) {

for (int i = 0; i < 1024; i++ ) {

array[i] = (array[i] \* factor) + offset

}

}

\_\_m128 factor4 = \_mm\_set1\_ps(factor);

\_\_m128 offset4 = \_mm\_set1\_ps(offset);

for ( int i = 0; i < 1024; I = i+ 4 ) {

\_\_m128 array4 = \_mm\_loadu\_ps(&(array[i]));

\_\_m128 product =\_mm\_mul\_ps(array4, factor4);

\_\_m128 result = \_mm\_add\_ps(product, offset);

\_mm\_storeu\_ps(&(array[i]), result);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* code segment 2 \*/

float sum(float \* a, int size) {

float sum = 0.0;

for ( int i = 0; i < size; i++ ) {

sum = sum + a[i];

}

return sum;

}

\_\_m128 sum4 = \_mm\_setzero\_ps();

int remainder = size % 4;

int i;

for ( i = 0; i < size - remainder; i = i + 4 ) {

\_\_m128 a4 = \_mm\_loadu\_ps(&(a[i]);

sum4 = \_mm\_add\_ps(sum4, a4);

}

// add the four partial sums

float temp[4];

\_mm\_storeu\_ps(temp, sum4);

float sum = temp[0]+temp[1]+temp[2]+temp[3];

// add remainder iterations

for ( ; i < size; i++ ) {

sum = sum + a[i];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// in the following, size can have any positive value

float routine\_3(float \* restrict a, float \* restrict b, int size) {

float sum\_a = 0.0;

float sum\_b = 0.0;

for ( int i = 0; i < size; i++ ) {

sum\_a = sum\_a + a[i];

sum\_b = sum\_b + b[i];

}

return sum\_a \* sum\_b;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 4 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// in the following, size can have any positive value

void routine\_4(float \* restrict a, float \* restrict b, int size) {

for ( int i = 0; i < size; i++ ) {

a[i] = 1.5379 - (1.0/b[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 5 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// in the following, size can have any positive value

void routine\_5(float \* restrict a, float \* restrict b, int size) {

for ( int i = 0; i < size; i++ ) {

if ( a[i] < b[i] ) {

a[i] = b[i];

}

}

}

int remainder = size % 4;

int i;

for ( i = 0; i < size - remainder; i = i + 4 ) {

\_\_m128 a4 = \_mm\_loadu\_ps(&(a[i]);

\_\_m128 b4 = \_mm\_loadu\_ps(&b[i]);

\_\_m128 max4 = \_mm\_max\_ps(a4, b4);

\_mm\_storeu\_ps(&(a[i]), max4);

}

for ( ; i < size; i++ ) {

if (a[i] < b[i] ) a[i] = b[i];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 6 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void multiply(float \*\* matrix, float \* vec, float \* result)

{

for ( int i = 0; i < 4096, i++ ) {

float sum = 0.0;

for ( int j = 0; j < 4096; j++ ) {

sum += vec[j] \* matrix[i][j];

}

result[i] = sum;

}

}

\_\_m128 sum4 = \_mm\_setzero\_ps();

for ( int j = 0; j < 4096; j = j + 4 ) {

// sum += vec[j] \* matrix[i][j];

\_\_m128 vec4 = \_mm\_loadu\_ps(&(vec[i]);

\_\_m128 mat4 = \_mm\_loadu\_ps(&(matrix[i][j]));

\_\_m128 product4 = \_mm\_mul\_ps(vec4, mat4);

sum4 = \_mm\_add\_ps(sum4, product4);

}

// sum the four partial products

float temp[4];

\_mm\_storeu\_ps(temp, sum4);

result[i] = temp[0]+temp[1]+temp[2]+temp[3];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 7 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// hint: one way to vectorize the following code might use

// vector shuffle operations

void routine\_7(float \* restrict a, float \* restrict b,

float \* restrict c) {

for ( int i = 0; i < 2048; i = i+2 ) {

a[i] = b[i]\*c[i+1] + b[i+1]\*c[i];

a[i+1] = b[i]\*c[i] - b[i+1]\*c[i+1];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 8 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// in the following, size can have any positive value

int routine\_8(unsigned char \* restrict a, unsigned char \* restrict b, int size) {

for ( int i = 0; i < size; i++ ) {

if ( a[i] != b[i] )

return 0;

}

return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* routine 9 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void routine\_6(float \* restrict a, float \* restrict b, float \* restrict c) {

a[0] = 0.0;

for ( int i = 1; i < 1023; i++ ) {

float sum = 0.0;

for ( int j = 0; j < 3; j++ ) {

sum = sum + b[i+j-1] \* c[j];

}

a[i] = sum;

}

a[1023] = 0.0;

}