Hardware für Eingebettete Systeme

Lab 2: GCC Vector Datatypes

Prof. Dr. Mario Porrmann
M.Sc. Marc Rothmann

1 Vector Datatypes

In the last lab, you have seen how gcc automatically generates vectorized code. Now we will introduce vector datatypes to have more control over vectorization. In GCC we can define a vector like this:

```
float __attribute__((vector_size(16), aligned(32))) vec;
```

This will create a vector of 4 floats. To make our lives easier, we can use a `typedef` if we want to use multiple vectors:

```
typedef float __attribute__((vector_size(16), aligned(32))) v4f;
v4f a = {1,2,3,4};
printf("%f\n", a[0]);
```

The above example also demonstrates how to initizalize a vector and how to access single elements. For simple math, you can simply use the usual C operators and they will be used on all the elements of a vector. Write a small test program that creates two vectors and performs vector addition and element-wise multiplication.

2 Matrix Computations

Open the matrix.c file and have a look at the code. It contains the following functions to handle matrices:

```
void matrixAlloc(Matrix *m, int nc, int nr);
void matrixClear(Matrix *m);
void matrixAddSIMD(Matrix *a, Matrix *b, Matrix *c);
void matrixMultSeq(Matrix *a, Matrix *b, Matrix *c);
void matrixMultSIMD(Matrix *a, Matrix *b, Matrix *c);
void matrixRandom(Matrix *m);
void matrixPrint(Matrix *m);
```

Most of the code is simple C code. In the matrixAlloc function, the `posix_memalign` function is used, which allocates aligned data on the heap.
The main function allocates a few matrices and executes a matrix addition, as well as a sequential and SIMD matrix multiplication. Furthermore, the main function contains code that can be used to benchmark the matrix multiplication.

Hint: For the following exercises, it is only necessary to change the `matrixAddSIMD` and `matrixMultSIMD` functions. However, you are allowed to make changes to the rest of the code.

a) Implement the `matrixAddSIMD` function using vector datatypes.

b) Implement the `matrixMultSIMD` function using vector datatypes.

c) (optional) set `run_benchmark` to 1 and run the benchmark. Use the `eval.py` script to plot the data. How does your optimized matrix multiplication compare to the baseline? (The python script uses the *pandas* and *matplotlib* libraries, which you might need to install.)