Lab Session 2 –Simple SIMD Programming Example

$$Z(n) = X(n) * Y(n), n = 0,\ldots, N-1$$

Basic case: X,Y and Z are Q15 real numbers
Complex case: X,y, and Z are Q15+jQ15 (complex Q15)

What to do

- Mandatory components
  - Write the basic SIMD code for SSE4 (128-bit SIMD) and AVX2 (256-bit SIMD), AVX512 if possible
  - Use SIMdE for ARM
  - Use Godbolt(use whatever it's FAST)
  - Time the routine with respect to a scalar implementation .This can be done by running the routine on the same data set thousands of times and compute the CPU time at the beginning and end.
  - Evaluate the speed-up for scalar->128-bit->256-bit
  - Extra optimizations
  - Explore compiler optimization
- Optional (bonus)
  - Complex numbers
- Google : online Intel Intrinsics guide, ARM Neon intrisics

Profiling

- Three “tools”
  - Generating assembly output with gcc
    - https://stackoverflow.com/questions/137038/how-do-you-get-assembler-output-from-c-c-source-in-gcc/48426040
  - Callgrind (valgrind)/Perf
  - Embedding timing functions into your program
- Callgrind
  - Tool to count cycles in a program and provide statistics for all functions used by a program
    - Does not provide insight about memory issues (need other tools like Vtune for this)

Callgrind Usage

- Callgrind is invoked as

```
valgrind --tool=callgrind PROGRAM
kcachegrind callgrind.out.*
```

- `Kcachegrind` provides a graphical characterization of the cycle-usage statistics of your program

Counting cycles in a program

- You can also add instructions in critical portions of your code using the supplied "time_meas.h" include file
- This can be done using standard timers or cpu clock tick counters
- The .h file provides three routines you can use to auto-profile your code
  - `reset_meas(time_stats_t *ts)` : reset a time stats variable ts
  - `start_meas(time_stats_t *ts)` : start a measurement ts
  - `stop_meas(time_stats_t *ts)`: stop a measurement ts