4 (De)compression using on-Chip Vector Processing Units

In main memory column-store databases the columns are often compressed in order to save RAM and to improve access rates. In this assignment you will speed up *decompression* of column values by exploiting data level parallelism in modern CPUs. In particular, you will be using *Streaming SIMD Extensions* (SSE) or Advanced Vector Extensions (AVX) if your system supports it, *Single Instruction, Multiple Data* (SIMD) instruction set extensions to the x86 architecture, originally designed by Intel.

4.1 SSE and AVX Intrinsics

To make use of SSE you do not need to write inline assembly code. GCC as well as Microsoft and Intel's C compilers implement special intrinsics[1] that map to the x86 SIMD instructions. As opposed to inline assembly, intrinsic functions can be fully accounted for by the compiler when optimizing your program. The C intrinsic functions and SSE or AVX assembly instructions are documented in the following manuals: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html. Especially interesting are volumes 2A and 2B. At the end of volume 2B you can find a list of all available intrinsics.

Moreover, "**Intel Intrinsic Guide**", an interactive GUI tool that allows searching and looking up of Intel's SIMD instructions. The guide also provides descriptions for each instruction, data type support as well as the intrinsic mnemonics. This tool would be very helpful while working on this assignment and highly suggested to be used by everyone. The tool can be downloaded from the following website: http://software.intel.com/en-us/avx

4.2 Warm-up Exercise

To get you started we have prepared a warm-up exercise for you that should help you get familiar with the SSE intrinsic functions. Please go to the course website and download the skeleton C code: http://www.systems.ethz.ch/sites/default/files/file/dpmh_Fall2012/src-handout-04_tar.bz2

Open the file `compareandcount.c`. You will find a simple C program that does the following:

1. An array is populated with 100 million numbers chosen randomly from the interval [0,99].
2. Then the program counts how many values in that array are larger than 42.
3. Finally, execution time for the computation is measured and displayed.

Your job is to speed up the scanning and counting of values > 42 as much as possible using SSE intrinsics. The relevant header files (`xmmintrin.h` $\rightarrow$ SSE, `tmmintrin.h` $\rightarrow$ SSSE3, `smmintrin.h` $\rightarrow$ SSE4.1) are already included. When you compile your code do not forget the flags `-mssse3 -msse4` and use an optimization level $\geq$ `O1` (or simply compile with the provided `Makefile`).

[1] Intrinsic functions are functions whose implementation is handled specially by the compiler, avoiding the overhead of a function call and allowing highly efficient machine instructions to be emitted for that function.

4.3 Compression in Column Stores

Now that you have completed the warm-up exercise you should be ready to implement SIMD - accelerated column decompression. This part of the exercise is based on the n-bit fixed-width *frame of reference* (FoR) compression that was shown in the lecture and is discussed in [1]. Nevertheless, we ignore the base value min_C of a column and are only interested in decompressing n-bit fixed-width values into 32-bit integers.

Task

In the archive that you have previously downloaded for this exercise you will find a second skeleton C code file: `compression.c`. In this file we have implemented compression and *serial* decompression for the following formats: 32-to-8, 32-to-9 and 32-to-7 bin compression. The default is the simple 32-to-8 bit compression. You can select the other compression variants using the respective C macros : `COMPRESS32TO7`, `COMPRESS32TO9`.

Your task in this exercise is to speed up decompression using SSE intrinsic functions. For that purpose, please implement the following function bodies in the C program:

1. `SIMD_decompress8to32(...)`
2. `SIMD_decompress9to32(...)`
3. `SIMD_decompress7to32(...)`

You can activate these functions using the `SIMD` macro provided in the code. While 32-to-8 bit decompression is rather straightforward 32-to-9 and 32-to-7 can become quite tricky since the values are no longer aligned on byte boundaries. Please refer to [2]—on which this assignment is based—for further insight on SIMD decompression techniques.

Note: If the speed-up that you measure is below your expectations, consider not storing the decompressed values back to memory, *i.e.*, modify the program such that the values are only read and decompressed, *e.g.*, to compute some aggregate (SUM for instance). Of course, the validation code will then break but by then you should be confident that your decompression implementation works appropriately.

References

[1] J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing relations and indexes. *Data Engineering, International Conference on*, 0:370, 1998.

[2] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, Alexander Zeier, and Jan Schaffner. Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units. *PVLDB*, 2(1):385–394, 2009.