UDC 519.245; 519.674; 004.032.24:004.272

P.O. Prystavka, DsC, A.K. Shevchenko
(National Aviation University, Ukraine)

A brief overview of modern software optimization approaches

A brief overview of modern software optimization approaches shows that last tech items (such as libraries and frameworks) are based on SIMD/MIMD instructions of CPU. The overview of SIMD/MIMD usage is provided for most popular software products like library's, frameworks and popular OS.

Acceleration by means of hardware.

Well chosen hardware architecture may significantly enhance software product performance. A general perspective on possible options is given by the Flynn's taxonomy (Tab. 1) and the first question to answer is whether the task (e.g. convolution operation) allows parallelization of data flow or instructions flow [1].

Table 1.

Flynn's taxonomy

Data \ Instructions	Yes	No
Yes	MIMD	SIMD
No	MISD	SISD

Today SIMD principles are implemented in both RISC (e.g. Cortex-A8–23 and Cortex-A53–72 ARM CPUs) [2] and CISC (e.g. Intel x86 series) CPUs. One can access this feature with specific extensions of assembly language – NEON32 and NEON64 for RISC architecture, while CISC architecture implies SSE_n and $AVX_{1/2}$ usage. In contrast, MIMD principles are not implemented in modern CPUs, but partially supported by GPUs.

Modern GPUs provide parallel computing features by means of shader blocks-CPU (SCPU), based on RISC architecture, that are used in parallel. Principles of MIMD are achieved by SIMD/MIMD-like instructions for SCPU – vector instructions for numerous 128/256/...-bit registers (there are 32 or more registers, that is above the number that modern CPUs have). The bad part is you cannot access SIMD/MIMD instructions directly – only a small number of intrinsics and pre-implemented operations are accessible: bit shifts, binary logic, etc. Most cases programmers use specific frameworks to access mentioned features, e.g. CUDA and OpenCl for GPU, or OpenCl for CPU/DSP/FPGU. Prevalence of mentioned frameworks led to most GPU manufacturers get certified by AMD/ATI (compatibility with OpenCl) or Intel/NVidia (compatibility with CUDA).

Except using GPU, one can employ co-processor units, e.g. Digital Signal Processor (DSP). For example, Qualcom has developed DSP-Hexagon for embedding into Snapdragon-625/635/835/825 CPUs. This DSP provides very long instruction word (VLIW), which means multithreading at the assembler level – during one

interruption 3 assembly instructions with different inputs are processed. Compared to simple SIMD (NEON32 or NEON64) its performance is 4 times higher. Algorithms, optimized for DSP, reduce CPU load up to ~ 75% and improve audio/video encoding/decoding performance up to ~ 18 times [3, 4].

Optimization by means of software.

Well Software we use to produce binary code (e.g. compiler itself, additional libraries, frameworks) highly influences program performance by employing different optimizations and hardware platform capabilities. In scope of current article we are mostly concerned with their ability to perform vectorization without significant loss in precision. Let's consider three well-known compilers: GNU Compiler Collection (GCC/G++) [5], Clang [6], and nvcc (compiles cu-files for CUDA).

Probably, the most popular nowadays is GCC developed and supported by FSF community. Actually, GCC, first developed by Richard Stallman, is a whole collection of compilers suitable for different programming languages and architectures.

Its main competitor is the "rising star" of compilers – Clang. For example, Apple already uses it as the basic compiler for its products. Clang itself is a frontend for different programming languages, e.g. C, C++, Objective-C, Objective-C++, and OpenCL. Actual generation of binary code and vectorization is performed by the LLVM framework.

Both Gcc and Clang are performance-oriented, but still they fail compared to human-made assembly code (see comparison [7] Clang ndk-r14b vs Inline Assembly on Android 5.5.1 (x64) phone with CPU-MT6752).

The last compiler we want to mention is nvcc. It utilizes CUDA and thus allows significant improvement of performance on platforms with NVidia GPU.

To avoid this large heterogeneity, OpenCl standard was developed (The Khronos Group Inc.) that is supported by all mentioned hardware developers and provides access to parallel computations on GPU/DSP/CPU.

Except all the advantages of PCPs, they have a drawback – big overhead on transferring data. To avoid the problem, programmers organize data into pools 100 ~ 200 MB, that allows to achieve 20-fold increase in performance compared to CPU. But using big pools is not always the solution − while CNN learning perfectly fits in this model, processing stream from video-camera does not at all. Except for good choice of compiler, one can achieve performance enhancement using optimized binary code of frequently used functions like CO, scaling, etc. supplied by different libraries. Many of them contain SIMD-optimized code for armeaby-v7a and arm64-v8a. Besides, a collection of libraries can be combined into a single framework in such a way, that advantages of one library compensate drawbacks of the others. OpenCV and ACL [9, 10] are good examples of libraries comprising a wide variety of algorithms, including DI processing, DI analysis, and even module for CNN learning, that are optimized for different CPU architectures and their SIMD: $AVX_{1/2}$, $SSE_{4.4}$, and ARM $NEON_{x32/x64}$. OpenCV is well-known and of a high quality, but ACL has better extensibility due to modular architecture and seems to perform better on CO-like tasks (e.g. it is up to ~14 times faster compared to OpenCV on CO for CNN in one thread [8, 9]). Thus, further we will use both of them as a reference for comparison.

At the moment SIMD optimization has spread over the wide range of programming products, both proprietary and open-source. For example, kernel of Windows 10 uses $AVX_{1/2}$ to achieve better performance (obviously, this influences the whole system), while Oracle Java VM utilizes $AVX_{1/2}$/3DNow and thus any Java application runs faster. Game engines of id Tech 2-4 (e.g. the one used in Quake III Arena) are good example of open-source projects with SIMD optimization. But, using SIMD, they all face the issue of translating floating-point code to fixed-point with acceptable loss in precision. This can be quite complicated, thus SIMD optimizations used in proprietary software are mostly non disclosable.

In conclusion, modern hardware provides mechanisms for vectorization, i.e. SIMD technologies, that can be used by programmers to enhance performance of the application. Most cases this technology is utilized by compiler to generate binary code without participation of programmer. Suitable choose of library may be handy as well – many libraries contain SIMD-optimized code. But in some cases human intervention is needed to get the most optimal result. Developing assembly code, one should represent the function in suitable for SIMD optimization form. It is not always possible and often some restrictions should be satisfied.

References

1. M. J. Flynn, Proceedings of the IEEE 54, 1901 (1966).
2. "Arm Cortex–A53 mpcore processor: Reference book of Cortex-A53 cpus," http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500g/DDI0500G_cortex_a53_trm.pdf (2013), [Online; accessed 27-November-2017].
3. "Qualcomm extends hexagon dsp," http://pages.cs.wisc.edu/~danav/pubs/qcom/hexagon_microreport2013_v5.pdf (2013), [Online; accessed 27-November-2017].
4. "Qualcomm hexagon dsp: An architecture optimized for mobile multimedia and communications," https://developer.qualcomm.com/download/hexagon/hexagon-dsp-architecture.pdf (2013), [Online; accessed 27-November-2017].
5. A. Griffith, GCC: the complete reference (McGraw-Hill, Inc., 2002).
6. B. C. Lopes and R. Auler, Getting started with LLVM core libraries (Packt Publishing Ltd, 2014).
7. П. Приставка and А. Шевченко, Актуальні проблеми автоматизації та інформаційних технологій , 78 (2016).
8. "Presentation of arm-cl," https://community.arm.com/graphics/b/blog/posts/arm-compute-library-for-computer-vision-and-machine-learning-now-publicly-available (2017), [Online; accessed 24-May-2017].
9. "Video presentation for arm-cl," https://developer.arm.com/technologies/compute-library?_ga=2.909169.