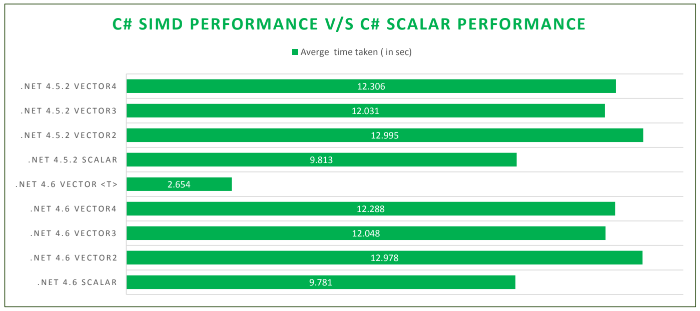
Performance per Watt : SIMD in C# & C++

Author: Rohan Verma - Systems Programmer, InfoSec | Test bench : Core i5 4960 Desktop processor, 16 GB DDR3

SIMD Performance Metrics for C#

Data for Matrix Multiplication performance (matrix size 100K x 100K) of different SIMD enabled datatypes in C#									
.NET Version	Platform	Type	Size	Running time #1	Running time #2	Running time #3	Running time #4	Running time #5	Averge time taken (in sec)
.NET 4.6	**x64**	**.NET 4.6 Scalar**	**1**	**00:00:09.5782544**	**00:00:09.8000870**	**00:00:09.8303289**	**00:00:09.8510545**	**00:00:09.8449726**	**9.781**
.NET 4.6	x64	.NET 4.6 Vector2	2	00:00:13.0134540	00:00:12.9750948	00:00:12.9497949	00:00:12.9902363	00:00:12.9635905	12.978
.NET 4.6	x64	.NET 4.6 Vector3	3	00:00:12.0862607	00:00:12.0503651	00:00:12.0494968	00:00:12.0197678	00:00:12.0331936	12.048
.NET 4.6	x64	.NET 4.6 Vector4	4	00:00:12.2995950	00:00:12.2592831	00:00:12.3159739	00:00:12.3360977	00:00:12.2297260	12.288
.NET 4.6	**x64**	**.NET 4.6 Vector <T>**	**8**	**00:00:02.6473882**	**00:00:02.6430108**	**00:00:02.6474216**	**00:00:02.6464199**	**00:00:02.6833316**	**2.654**
.NET 4.5.2	x64	.NET 4.5.2 Scalar	1	00:00:09.8337974	00:00:09.8645474	00:00:09.7548781	00:00:09.7829720	00:00:09.8267084	9.813
.NET 4.5.2	x64	.NET 4.5.2 Vector2	2	00:00:13.0383896	00:00:12.9730745	00:00:12.9807081	00:00:13.0018747	00:00:12.9816806	12.995
.NET 4.5.2	x64	.NET 4.5.2 Vector3	3	00:00:12.0389820	00:00:12.0227552	00:00:12.0345622	00:00:12.0262596	00:00:12.0330201	12.031
.NET 4.5.2	x64	.NET 4.5.2 Vector4	4	00:00:12.3170185	00:00:12.3023854	00:00:12.3078823	00:00:12.2871169	00:00:12.3158911	12.306

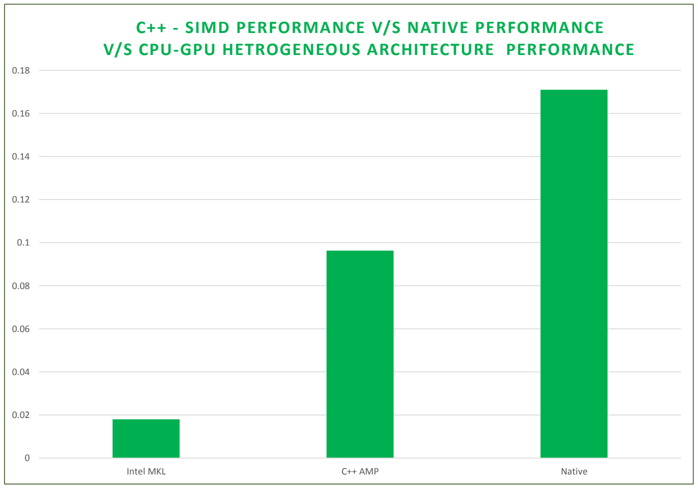

C# SIMD PERFORMANCE V/S C# SCALAR PERFORMANCE

- In the SIMD performance chart given, the less the time taken by the datatype would mean higher performance.
- An interesting insight is that in both .NET 4.6 & .NET 4.5 the performance of Vector2 , Vector3 and Vector 4 is significantly lower than Scalar type.
- Vector<T> datatype has highest performance, this datatype has only true vectorization enabled.
- Because the Vectorization is enabled for Vector<T> only, the datatype such as Vector2, Vector3 and Vector4 has no vectorization because they are fixed size types.
- C# has inherited concept of templates in datatypes (Vector <T>) from C++ and despite of draw-backs of templates there is no negative impact on JIT compiler and performance of the datatype.

SIMD Performance Metrics for C++

Data for Matrix Multiplication performance (matrix size 100 x 100) of SIMD code & non-SIMD code in C++										
Library	Running Time	is SIMD ?	Platform	Compiler	SIMD instructions	Threads	System load	GPU enabled code	DRAM performance	Cache Sentitivity
Intel MKL	0.018	Yes	x64	Intel C++ '17	SSE/SSE2	1(CPU thread)	Random	No	High	L1- cache sensitive
C++ AMP	0.09632	No	x64	Visual Studio 15 C++	N/A	>1 (GPU threads)	Random	Yes	High	N/A
Native	0.171	No	x64	Visual Studio 15 C++	N/A	1(CPU thread)	Random	No	High	L2- cache sensitive

- The Matrix size in case of C++ is 100 x 100 not 100K x 100K because the array size can't be too high. There is a limitaion defined even in case of STL Vectors.
- C++ SIMD performance for the Intel MKL or Intel Math Kernel is dominant because it has significant least running time than others. The analysis shows that Intel MKL implementaion is very sensitive to L1 cache stalls/miss because we are using datatypes which are 128 bit wide. L1-cache or L1- Data cache is 32KB in size thus it can't hold more than 2K such datatypes , more specifically cache-lines have to be replaced which in turn will produce stalls. Non-contiguous memory allocation by allocator increases stalls.
- For achieving higher performance in matrix multiplication the library called C++ AMP is used. This platform independent library makes the use of CPU-GPU heterogeneous architecture to do matrix multiplication, the results shows that CPU/GPU heterogeneous architecture is still slower than Intel's Intrinsic SIMD.
- Intel MKL uses SIMD intrinsic datatypes like __m128 to produce the result. The intel C++ compiler also provides auto-vectorization which was not available in .NET environment. So even if the data doesn't use datatype which are SIMD specific, the code or instruction can be parallelized for higher throughput.


C++ - SIMD PERFORMANCE V/S NATIVE PERFORMANCE V/S CPU-GPU HETROGENEOUS ARCHITECTURE PERFORMANCE