Arm Solutions at Lightspeed

Benchmarking Clang on Windows Arm using SPEC CPU 2017

Author - Omair Javaid

Speaker - Maxim Kuvyrkov

About SPEC CPU 2017

- Benchmark suite for compute-bound performance comparison.
- Measures integer and floating-point performance
- **SPECrate (multi-threaded)** Measures system throughput by running multiple threads.
- **SPECspeed (single-threaded)** Evaluates single-thread performance by running one task at a time

Key Focus Areas of SPEC CPU 2017

- CPU and memory architecture performance
- Impact of compiler optimizations on performance
- Not designed to test I/O, graphics, or networking performance.
- Measures how well a compiler and processor work together for compute-intensive applications.

Set Up SPEC CPU 2017 on Windows

Machine Preparation

- It is recommended to start with a fresh install of Windows for a clean environment.
- Avoid unnecessary software installations to minimize performance interference.
- Ensure all system updates are installed to avoid interruptions.
- Disable automatic updates and restarts during the benchmark run.

Install Software Dependencies

- Install Visual Studio with MSVC tools for ARM64 or x64.
- Install 7zip, LLVM, Make, CMake, Python, Ninja, Perl
- Ensure all tools are correctly installed and added to the system PATH.

Set Up SPEC CPU 2017 on Windows

SPEC CPU 2017 Configuration on Windows

- Obtain and Install SPEC CPU 2017. Unzip the benchmark files and **run install.bat** successfully.
- Add **<SPEC>\bin** to the system's environment path.
- Set MSVC development environment by running **vcvarsarm64.bat** or **vcvarsamd64.bat**
- **<SPEC>\config** folder has sample configuration files for visual studio
- Update **Example-VisualStudio.cfg** for clang-cl and cl.exe.
- Set architecture and optimization flags (/Od, /Ot, /Os, /O1, /O2).
- Set the number of **copies** in the config file to run multiple instances of benchmark in parallel
- Set the number of **threads** in the config file to run multiple threads for every benchmark instance.
- Use runcpu.bat from **<SPEC>\bin\windows** to execute benchmarks

Test Environment & Configurations

Hardware

- Microsoft Surface Pro - 11th Edition
- Snapdragon(R) X 12-core X1E80100 3.40 GHz
- 32 GB RAM - 1 TB SSD

Software

- Windows 11 Pro 24H2 - Build 26100.1742
- LLVM 19.1.0 for Windows on Arm
- Microsoft Visual Studio 2022
- MS C/C++ Optimizing Compiler Version 19.41.34120 for ARM64

Tested Configurations

Clang-cl

- -march=armv8.7 /Os
- -march=armv8.7 /Ot
- -march=armv8.7 /Od

MSVC cl

- /arch:armv8.7 /O1
- /arch:armv8.7 /O2
- /arch:armv8.7 /Od

SPEC CPU 2017 Build on Windows

- Builds ran at 26 degree celsius room temperature.
- clang-cl builds and run all benchmarks successfully with **no_fortran**
- **Flang-new** still fails to build SPEC benchmarks on windows
- **510.parest_r** test fails to build with MS cl compiler

SPEC CPU 2017 Execution Time (seconds)						
	cl /Od	**clang-cl /Od**	**cl /O1**	**clang-cl /O1**	**cl /O2**	**clang-cl /O2**
Total	26057	26259	10318	9958	9423	9714
parest_r	0	1792.94	0	376.74	0	350.25

Execution Times: clang-cl vs cl

SPEC Rate INT

- **clang-cl** consistently performs better than cl across all three optimization levels.
- The highest **improvement** is in the **x264_r benchmark**
- xalancbmk_r and xz_r show slight performance degradation with clang-cl undre /O1

	/O1			/O2			/Od		
Benchmark	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**
500.perlbench_r	270.51	245.8	9.13	270.37	236.32	12.59	605.07	583.56	3.56
502.gcc_r	197.67	170.46	13.77	179.58	167.56	6.69	465.9	457.68	1.77
505.mcf_r	251.23	248.01	1.28	236.22	239.91	**-1.56**	458.33	454.16	0.91
520.omnetpp_r	498.57	473.31	5.07	491.9	471.66	4.11	1083.36	1060.97	2.07
523.xalancbmk_r	215.65	217.16	**-0.7**	209.78	195.03	7.03	905.27	940.56	**-3.9**
525.x264_r	386.2	155.81	**59.63**	96.32	76.54	**20.55**	1261.19	1052.95	**16.5**
531.deepsjeng_r	218.53	193.90	11.27	208.64	192.62	7.68	543.82	518.99	4.56
541.leela_r	379.51	335.55	11.59	340.06	293.59	13.68	1587.45	1473.77	7.15
557.xz_r	351.98	356.35	**-1.24**	377.58	342.22	9.37	799.51	799.08	0.05

Execution Times: clang-cl vs cl

SPEC Rate FP

- **clang-cl** shows significantly improved execution times over all optimization levels.
- **lbm_r** shows a surprising improvement of 43.97% with clang-cl at /Od
- **magick_r** shows minimal improvement of 0.41% at /O2 with clang-cl.

	/O1			/O2			/Od		
Benchmark	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**
508.namd_r	117.44	96.57	17.77	115.13	96.52	16.17	618.37	515.71	16.6
511.povray_r	406.01	309.32	23.82	368.33	284.03	22.89	1045.76	957.84	8.41
519.lbm_r	235.42	203.9	13.39	213.82	203.29	4.93	454.93	254.88	**43.97**
526.blender_r	188.07	155.19	17.48	169.59	144.64	14.71	738.63	600.87	18.65
538.imagick_r	283.58	209.52	26.12	206.82	205.97	**0.41**	765	701.84	8.26

Execution Times: clang-cl vs cl

SPEC Speed INT

- **clang-cl** shows significantly improved execution times across most benchmarks.
- **x264_s** benefits the most from clang-cl at all optimization levels
- **gcc_s** and **xalancbmk_s** show slight regressions under /O2 and /Od, respectively.

	/O1			/O2			/Od		
Benchmark	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**
600.perlbench_s	243.04	195.35	19.62	242.02	191.95	20.69	506.43	505.05	0.27
602.gcc_s	393.34	325.19	17.33	322.91	334.62	**-3.63**	1076.58	998.68	7.24
605.mcf_s	394.45	374.04	5.17	371.02	362.2	2.38	716.4	721.09	-0.65
620.omnetpp_s	368.01	344.73	6.33	364	336.85	7.46	871.38	851.67	2.26
623.xalancbmk_s	179.61	174.62	2.78	170.74	169.35	0.82	800.55	820.18	**-2.45**
625.x264_s	344.85	139.4	**59.59**	109.04	65.5	**39.95**	1126.58	939.11	**16.64**
631.deepsjeng_s	251.51	225.03	10.54	240.51	223.55	7.06	608.54	579.6	4.75
641.leela_s	351.31	298.34	15.08	315.17	256.74	18.51	1416.38	1301.59	8.1
657.xz_s	808.13	788.44	2.44	793.19	770.84	2.82	1559.49	1522.28	2.39

Execution Times: clang-cl vs cl

SPEC Speed FP

- **imagick_s** exhibits execution time degradation with **clang-cl** under both /O1 and /O2
- **nab_s** build with clang-cl has significant improvements under /O1 and /O2
- **lbm_s** shows best performance upgrade under clang-cl at /Od

	/O1			/O2			/Od		
Benchmark	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**
619.lbm_s	370.13	337.67	8.77	365.61	360.8	1.32	460.54	353.6	23.22
638.imagick_s	844.58	919.55	**-8.88**	835.95	885.43	**-5.92**	2887.25	2782.79	3.62
644.nab_s	426.75	360.14	**15.61**	416.66	362.21	**13.07**	1151.27	1038.55	**9.79**

EXE File Size: clang-cl vs cl

- **clang-cl** generates significantly small sized executables.
- **clang-cl** executables are on average **40%** smaller across all optimization levels

	/O1			/O2			/Od		
Benchmark EXE	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**
perlbench_r.exe	3.81 MB	2.38 MB	-37.70%	4.07 MB	2.64 MB	-35.21%	6.27 MB	3.78 MB	-39.72%
cpugcc_r.exe	14.42 MB	8.02 MB	-44.40%	15.88 MB	8.98 MB	-43.45%	22.22 MB	13.39 MB	-39.71%
mcf_r.exe	0.92 MB	0.53 MB	-42.61%	0.92 MB	0.53 MB	-42.08%	0.94 MB	0.54 MB	-42.68%
namd_r.exe	2.12 MB	1.34 MB	-36.77%	2.30 MB	1.45 MB	-37.05%	3.04 MB	1.78 MB	-41.62%
povray_r.exe	2.27 MB	1.33 MB	-41.54%	2.44 MB	1.51 MB	-38.08%	3.10 MB	1.74 MB	-43.93%

PDB File Size: clang-cl vs cl

- **clang-cl** produces on average around 15% smaller PDB files across all optimization levels.
- **Blender_r** built with clang-cl at /Od exhibits a 21.58% increase in PDB size
- **clang-cl** seems to reduce more debug info at /O1 and /O2

	/O1			/O2			/Od		
Benchmark	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**	**cl**	**clang-cl**	**(%)**
xalancbmk_s.pdb	66.20 MB	36.97 MB	**44.15**	69.49 MB	37.71 MB	**45.72**	59.30 MB	50.32 MB	**15.14**
cpuxalan_r.pdb	66.17 MB	36.97 MB	**44.12**	69.46 MB	37.71 MB	**45.71**	59.28 MB	50.32 MB	**15.11**
sgcc.pdb	34.02 MB	24.70 MB	**27.39**	35.86 MB	25.62 MB	**28.57**	25.91 MB	24.54 MB	**5.29**
cpugcc_r.pdb	34.00 MB	24.70 MB	27.35	35.85 MB	25.62 MB	28.54	25.90 MB	24.54 MB	5.26
perlbench_s.pdb	13.52 MB	10.64 MB	21.29	13.75 MB	10.97 MB	20.22	12.37 MB	10.77 MB	12.91
imagevalidate_526.pdb	7.14 MB	5.70 MB	20.23	7.14 MB	5.70 MB	20.14	7.13 MB	5.66 MB	20.60
povray_r.pdb	11.71 MB	9.35 MB	20.12	11.90 MB	9.62 MB	19.17	10.83 MB	9.16 MB	15.40
imagick_r.pdb	13.12 MB	10.73 MB	18.19	13.36 MB	10.86 MB	18.69	11.80 MB	10.38 MB	12.08
blender_r.pdb	60.38 MB	55.94 MB	7.36%	62.57 MB	58.09 MB	7.15%	49.58 MB	60.28 MB	**- 21.58**

WindowsPerf

- https://github.com/arm-developer-tools/windowsperf
- A Linux perf inspired tool designed specifically for Windows on Arm platforms.
- Utilizes the ARM64 Performance Monitor Unit (PMU) and its hardware counters for profiling.
- Enables Deep Performance Analysis on Windows on Arm devices, similar to Linux environments.

How WindowsPerf with compiler optimizations?

- May find optimization opportunities with fine-grained profiling
- Provides detailed performance data at the function, basic block, and instruction levels.
- Analyze how applications interact with Windows-specific system calls, APIs, and kernel components.

Thank You!

Visit linaro.org

EMAIL: omair.javaid@linaro.org

GITHUB: https://github.com/omjavaid

DISCOURSE:
https://discourse.llvm.org/u/omjavaid/