Linaro Connect
MADRID 2024 | MAY 14-17 2024

ARM64 Optimizations in MSVC 2022

Hongyon Suauthai

Agenda

- Windows on Arm Introduction
- Arm64EC("Emulation Compatible")
- Optimizations in 17.6-17.10
- Current work and Future plan
- Conclusion

Windows on Arm (WoA)

- Lightweight
- Extends Battery life
- Powers on instantly
- Always connected to the internet

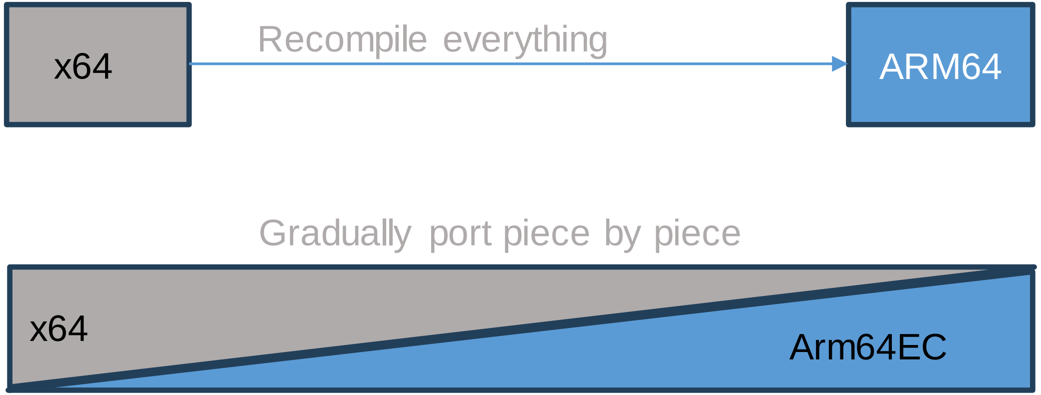
Lenovo ThinkPad X13s

Xiaomi Book S

Microsoft Project Volterra

Arm64EC(“Emulation Compatible”)

- New application binary interface(ABI)
- Build only performance critical components natively.
- Keep existing dependencies/plugins or arch-specific code while porting.



Arm64EC("Emulation Compatible")

- Easily transition to native apps on ARM64 from x64
- Seamlessly interoperate with x64 binaries.
- Windows 11 on Arm binaries:

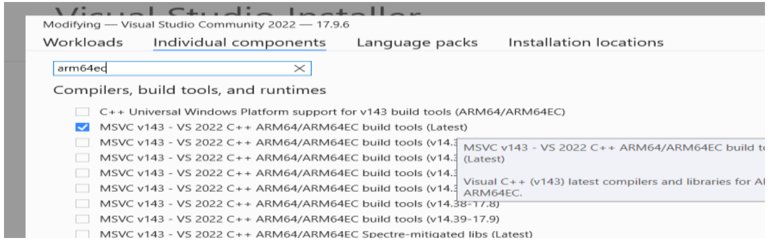
PE architecture	x64 lib	Arm64EC lib	Arm64 lib
Arm64EC	✓	✓	❌
Arm64	❌	❌	✓

Process architecture	x64 binary	Arm64EC binary	Arm64 binary
x64/Arm64EC	✓	✓	❌
Arm64	❌	❌	✓

✓ = Supported, ❌ = Not supported

Arm64EC(“Emulation Compatible”)

- Requirements
  - Windows 11 SDK Build
  - Visual Studio 2022
  - Arm64EC tools



Modifying — Visual Studio Community 2022 — 17.9.6

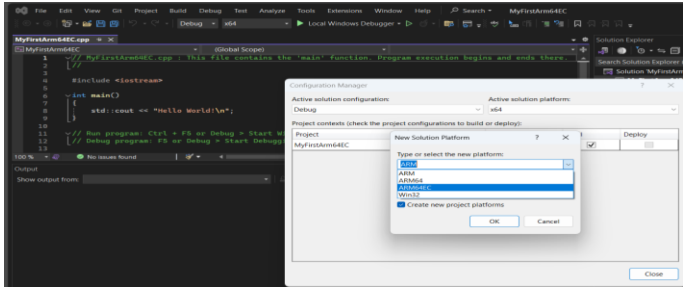
arm64ec

Compilers, build tools, and runtimes

- □ C++ Universal Windows Platform support for v143 build tools (ARM64/ARM64EC)
- ☑ MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools (Latest)
- □ MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools (v14.3
- □ MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools (v14.3
- □ MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools (v14.3
- □ MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools (v14.3
- □ MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools (v14.38-17.8)
- □ MSVC v143 - VS 2022 C++ ARM64/ARM64EC build tools (v14.39-17.9)
- □ MSVC v143 - VS 2022 C++ ARM64/ARM64EC Spectre-mitigated libs (Latest)

MSVC v143 - VS 2022 C++ ARM64/ARM64EC build t (Latest)

Visual C++ (v143) latest compilers and libraries for A ARM64EC.



Debug x64

```
// MyFirstArm64EC.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
}

// Run program: Ctrl + F5 or Debug > Start W
// Debug program: F5 or Debug > Start Debugg
```

Configuration Manager

Active solution configuration: Debug

Active solution platform: x64

Project contexts (check the project configurations to build or deploy):

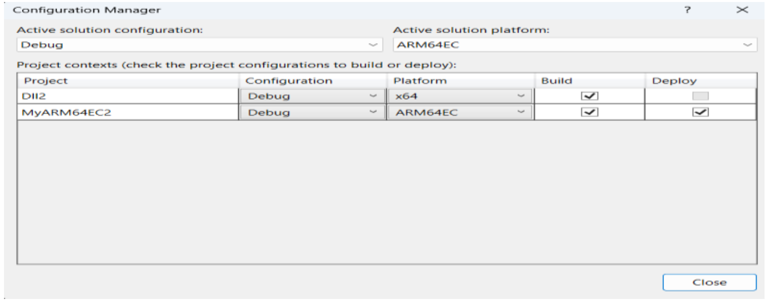
Project		Deploy
MyFirstArm64EC	☑	□

New Solution Platform

Type or select the new platform: ARM

- ARM
- ARM64
- ARM64EC
- Win32

☑ Create new project platforms



Active solution configuration: Debug

Active solution platform: ARM64EC

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
Dll2	Debug	x64	☑	□
MyARM64EC2	Debug	ARM64EC	☑	☑

MSVC Compiler Optimizations(17.6-17.10)

SIMD improvements:

- Supports more NEON instructions with asymmetric operands
- Supports small types on ABS/MIN/MAX
- Enables more by element operations
- Supports for shift right and accumulate immediate
  - USHR+ADD -> USRA
- Shift into cmp
- Right shift and narrow into shifted narrow
  - SSHR+XTN -> SHRN
- Auto-Vectorizer supports conversions between floating-point and integer
- Supports extended left shifts
  - SXTL+SHL -> SSHLL
- Improved libC runtime library

Scalar improvements:

- MOVI/MVNI
- Improve negation of bool value
- Eliminate redundant comparisons
- Improved on immediate materialization for CMP/CMN
- Improved on logic immediate loading
- Catches more CCMP opportunities
- Using MOVI/MVNI for immediate move in smaller loops

https://devblogs.microsoft.com/cppblog/msvc-arm64-optimizations-in-visual-studio-2022-17-6/
https://devblogs.microsoft.com/cppblog/msvc-arm64-optimizations-in-visual-studio-2022-17-7/
https://devblogs.microsoft.com/cppblog/msvc-arm64-optimizations-in-visual-studio-2022-17-8/

Auto-Vectorizer supports more SIMD instructions with asymmetric operands

Source code

```
void smlal(int * __restrict dst, int * __restrict a, short * __restrict b, short * __restrict c) {
  for (int i = 0; i < 4; i++)
    dst[i] = a[i] + b[i] * c[i];
}
```

Generated Code in MSVC 17.5	Generated Code in 17.6
sxtl v19.4s,v16.4h sxtl v18.4s,v17.4h mla v20.4s,v18.4s,v19.4s	smlal v16.4s,v17.4h,v18.4h

Supported: SADDL/UADDL/SSUBL/USUBL

Now support: SMLAL/UMLAL/SMLSL/UMLSL

SIMD supports for more by element operations

Source code

```
void test(float * __restrict a, float * __restrict b, float c) {
    for (int i = 0; i < 4; i++)
        a[i] = b[i] * c;
}
```

Generated Code in MSVC 17.6	Generated Code in 17.7
dup v17.4s,v0.s[0] ldr q16,[x1] fmul v16.4s,v17.4s,v16.4s str q16,[x0]	ldr q16,[x1] fmul v16.4s,v16.4s,v0.s[0] str q16,[x0]

MOVI/MVNI for immediate move in smaller loops

Source code

```
void vect_movi_msl (int *__restrict a, int *__restrict b, int *__restrict c) {
   for (int i = 0; i < 8; i++)
      a[i] = 0x1200;                    // 0x12 << 8 = 0x1200

    for (int i = 0; i < 8; i++)

      c[i] = 0x12ffffff;                // ~((0xED) << 0x18) = 0x12ffffff
}
```

Generated Code in MSVC 17.7	Generated Code in 17.8

|movi_msl| PROC
  mov   x9, #0x1200
  movk x9, #0x1200, lsl #0x20
  ldr   x8, |$LN29@movi_msl|
  stp   x9, x9, [x0]
  stp   x8, x8, [x2]
  stp   x9, x9, [x0, #0x10]
  stp   x8, x8, [x2, #0x10]
|$LN29@movi_msl|
  DCQ   0x12ffffff12ffffff
	|movi_msl| PROC
	  movi v17.4s, #0x12, lsl #8
	  mvni v16.4s, #0xED, lsl #0x18
	  stp q17, q17, [x0]
	  stp q16, q16, [x2]

Scalar code-generation now catches more CCMP

Source code

```
int test (int a)
{
   return a == 17 || a == 32;
}
```

Generated Code in MSVC 17.7	Generated Code in 17.8
cmp w0, #0x11 beq |$LN3@test\| cmp w0, #0x20 mov w0, #0 bne \|$LN4@test| |$LN3@test\| mov w0, #1 \|$LN4@test| ret	cmp w0, #0x11 mov w8, #0x20 ccmpne w0, w8, #4 cseteq w0 ret

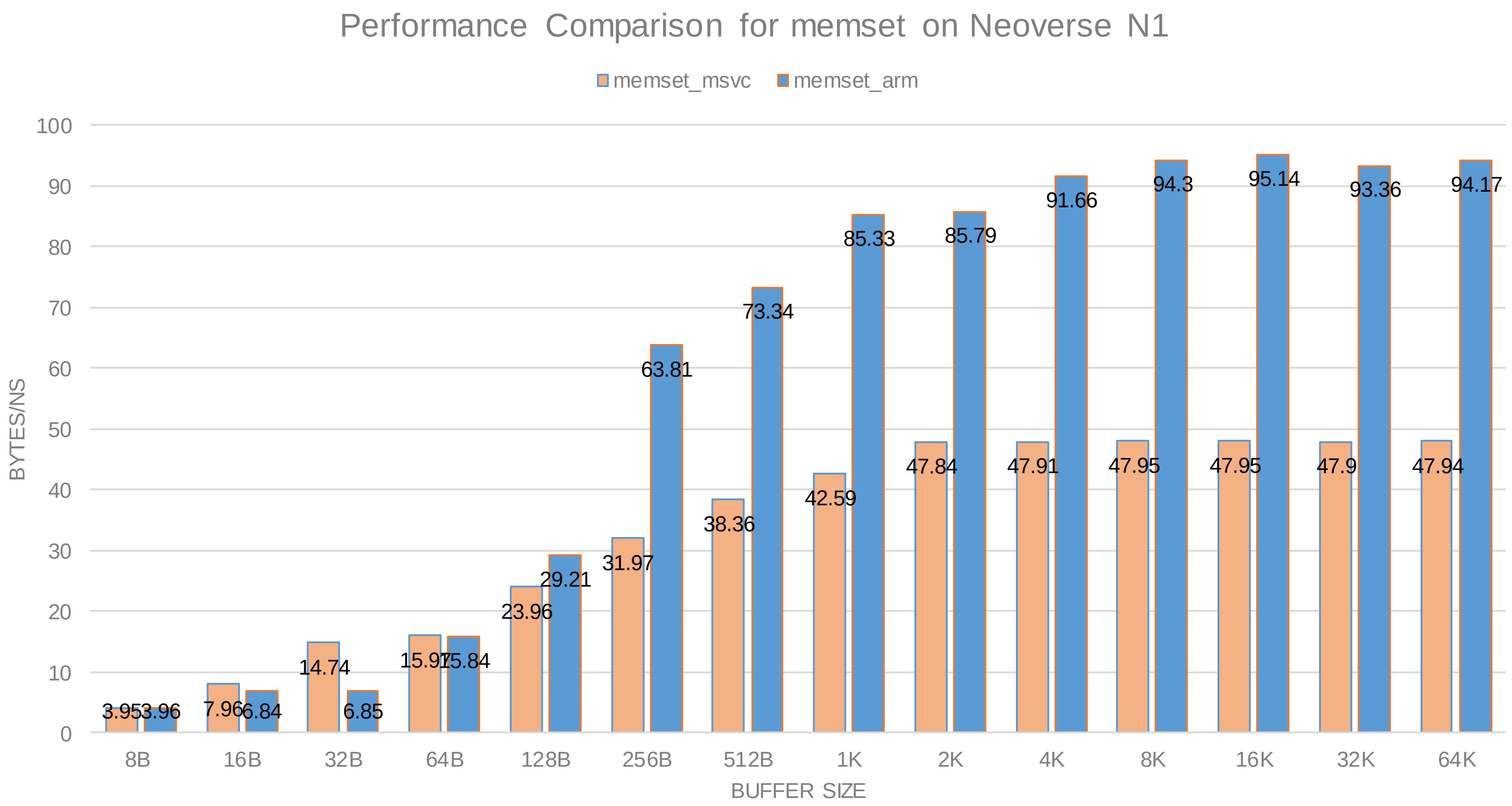
C runtime library optimization in 17.10

- Microsoft C Runtime Library(CRT)
  - UCRT
    - Standard C library
    - Conform close to ISO C99.
    - POSIX extensions
    - Microsoft-specific functions, macros, global variables
    - Part of Windows SDK
  - VCRUNTIME
    - Compile-specific runtime support library
    - Contains code required to support program startup
    - Features; exception handling, intrinsics
  - Link order:

    vcruntime then ucrt

- Following routines are optimized
  - memset, memcpy, memchr, memcmp
  - strlen, strchr, strrchr
- Integrated arm optimized routines from github
  - Use unaligned memory access
  - https://github.com/ARM-software/optimized-routines

- Inlined libC code generated will be available in VS 17.11


Performance Comparison for memset on Neoverse N1
memset_msvc
memset_arm
BYTES/NS
100
90
80
70
60
50
40
30
20
10
0
3.95
3.96
7.96
6.84
14.74
6.85
15.97
15.84
23.96
29.21
31.97
63.81
38.36
73.34
42.59
85.33
47.84
85.79
47.91
91.66
47.95
94.3
47.95
95.14
47.9
93.36
47.94
94.17
8B
16B
32B
64B
128B
256B
512B
1K
2K
4K
8K
16K
32K
64K
BUFFER SIZE

MSVC current plan and future work

- SVE/SVE2 - full assembly support in 17.10
- SVE intrinsic support in the work
- SME assembly support on the way
- SVE/SVE2 Auto-vectorization coming
- Security feature
  - Prevent ROP attack -> /guard:signret, based on PAC
  - Control Flow Guard: /guard:cf
  - BTI?

Conclusion

- MSVC continues to make performance improvements
- SVE/SVE2 support is coming
- VS developer community great way to give feedback, report bugs

  https://developercommunity.visualstudio.com/cpp
- Follow MSVC C++ blogs

  https://devblogs.microsoft.com/cppblog
- Arm64EC help links:
  - https://learn.arm.com/learning-paths/laptops-and-desktops/win_arm64ec/app_arm64ec/
  - https://on-demand.arm.com/flow/arm/devhub/sessionCatalog/page/pubSessCatalog/session/1681291098511001BlFX
  - https://devblogs.microsoft.com/cppblog/arm64ec-support-in-visual-studio/

Linaro Connect
MADRID 2024 | MAY 12-17 2024

Thank you