

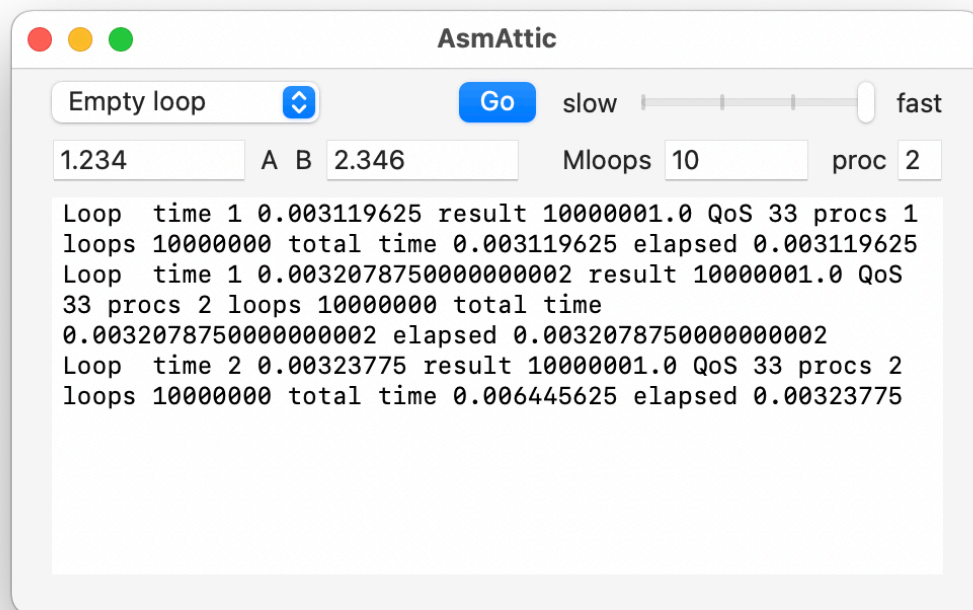
AsmAttic 4.5 for M1, M1 Pro and M1 Max

AsmAttic 4.5 is a utility for exploring performance of the different cores in Apple's M1 series of SoCs, as supplied in the following models of Mac:

- M1 MacBook Air 2020
- M1 MacBook Pro 13-inch 2020
- M1 Mac mini 2020
- M1 iMac 24-inch 2021
- M1 Pro MacBook Pro 2021
- M1 Max MacBook Pro 2021.

The app doesn't run *at all* on any Intel Mac, nor on any version of macOS before Big Sur 11.3.

Use



AsmAttic has a single window. When you close this, the app quits.

This window contains two rows of controls at the top, and a scrolling text view which fills the rest of the window and contains all results.

To run a benchmark test, select **which test** to run in the popup menu at the top left, set the slider at the right to determine its scheduling **QoS**, then enter values for **A**, **B**, the number of **Mloops** and the number of **processes**. When you're ready to run the test, click the **Go** button or press the Return key. After an interval, the result will appear in the scrolling view.

There are six different tests currently available in the popup menu:

- **Empty loop** runs an almost empty loop coded in assembly language.
- **Integer** runs arithmetic calculations using 64-bit integers, coded in assembly language.
- **Floating-point** runs the same arithmetic calculations using 64-bit floating-point values, coded in assembly language.

- **NEON** runs a dot product calculation on 32-bit floating-point vectors of length 4, coded in assembly language to run on the NEON SIMD unit.
- **Accelerate** runs a similar dot product calculation on 32-bit floating-point vectors of length 4, this time using the macOS Accelerate library.
- **Mixed** runs an inefficient mixture of integer and floating-point arithmetic, compiled from Swift.

The **QoS** slider determines the ‘Quality of Service’, with **fast** being the fastest, and **slow** the slowest. Normally on M1 chips, **slow** runs the test entirely on the Efficiency cores, and the other three settings use the Performance cores, and will also use the Efficiency cores as necessary.

A and **B** are starting values used to run the benchmark tests, and are floating point numbers. It’s simplest to start with relatively low values, around 1.0 or 2.0, but you can use positive and negative values much smaller or larger as you wish. In the dot product routines, for example, the A values are used to populate the vector **a**, and the B values for vector **b**. Each test then calculates the dot product

$$\mathbf{a} \cdot \mathbf{b} = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4$$

A third vector **c** is initialised with values of twice those in **a**. At the end of the dot product calculation, vector **a** is incremented by **c** ready for the next loop.

The **Mloops** value determines how many times the benchmark calculation is performed in each process. This is a floating-point number which is then multiplied by one million (1,000,000) and converted to an integer (64-bit) to set the number of loops to perform. Typical values range from 10 (for the Mixed test), to 1,000 (for most others) and 10,000 (for the empty loop).

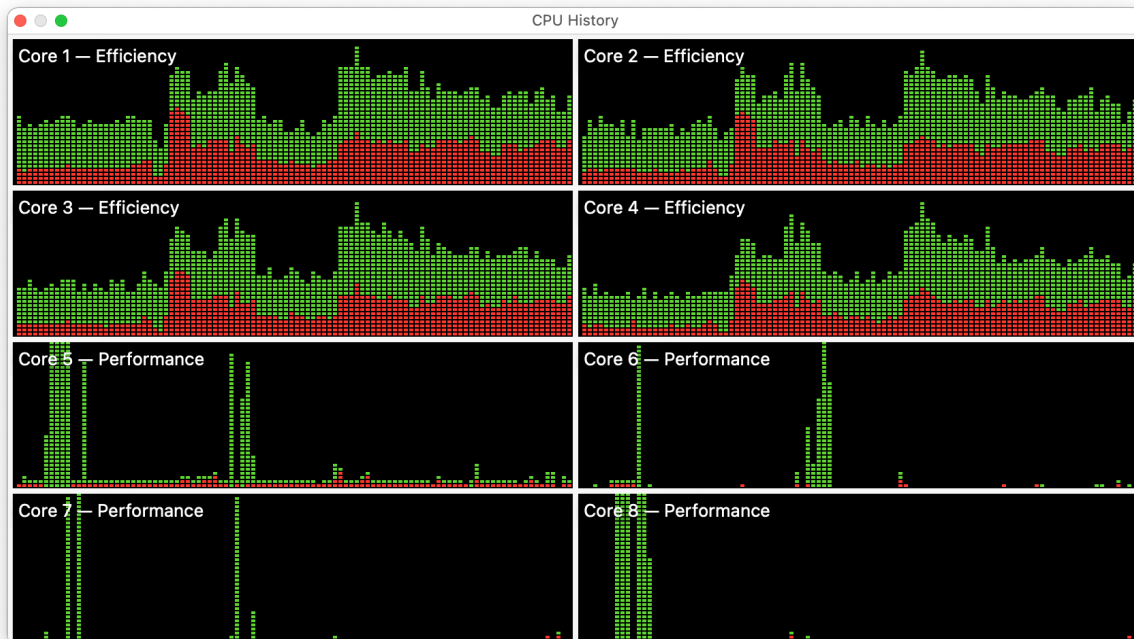
The **proc** value determines how many identical processes are run. For example, to fully occupy all the E cores on an original M1, set this to 4; on the M1 Pro/Max chip, 2 will occupy its two E cores. This number can range from 1 to 99.

Each test result reports:

- the test performed, e.g. Int for Integer
- the process number, which will be any value between 1 and the number of processes run, launched in numerical order, with 1 first
- the measured elapsed time taken by that process, in seconds
- the result, which should be the same for each test you perform for any given A, B and Mloops values
- the QoS setting for that process
- the total number of processes to be run
- the number of loops performed
- the total time for all completed processes so far, in seconds
- the overall elapsed time at the completion of that process, in seconds. The maximum, which should be given with the last process to complete, is thus the overall time to complete that test.

AsmAttic 4.5 for M1, M1 Pro and M1 Max

Every test is run in background threads using the control settings at the time the test was started using the Go button or Return key. You can easily set several tests running in rapid succession, using different QoS levels to have them run on different cores. Watch the effects in the CPU History view in Activity Monitor, to see which cores become loaded.



In this case, tests were run when the Efficiency cores were already loaded with a large Time Machine backup. Two tests were run at high QoS, as seen in the short peaks in load of the Performance cores, and two tests at low QoS, which are seen in the increased load of the Efficiency cores.

Tests

Source code is given here for each of the tests performed.

Empty loop

```
_emptyloop:
    STR    LR, [SP, #-16]!
    MOV    X4, X0
    ADD    X4, X4, #1
    MOV    X0, X1
mt_while_loop:
    SUBS    X4, X4, #1
    B.EQ    mt_while_done
    ADD    X0, X0, #1
    B       mt_while_loop
mt_while_done:
    LDR    LR, [SP], #16
    RET
```

Integer

```

_intmadd:
    STR    LR, [SP, #-16]!
    MOV    X4, X0
    ADD    X4, X4, #1
int_while_loop:
    SUBS   X4, X4, #1
    B.EQ   int_while_done
    MADD   X0, X1, X2, X3
    SUBS   X0, X0, X3
    SDIV   X1, X0, X2
    ADD    X1, X1, #1
    B      int_while_loop
int_while_done:
    MOV    X0, X1
    LDR    LR, [SP], #16
    RET

```

Floating-point

```

_fpfmadd:
    STR    LR, [SP, #-16]!
    MOV    X4, X0
    ADD    X4, X4, #1
    FMOV   D4, D0
    FMOV   D5, D1
    FMOV   D6, D2
    LDR    D7, INC_DOUBLE
fp_while_loop:
    SUBS   X4, X4, #1
    B.EQ   fp_while_done
    FMADD  D0, D4, D5, D6
    FSUB   D0, D0, D6
    FDIV   D4, D0, D5
    FADD   D4, D4, D7
    B      fp_while_loop
fp_while_done:
    FMOV   D0, D4
    LDR    LR, [SP], #16
    RET
INC_DOUBLE: .double    1.000000

```

NEON

```

_neondotprod:
    STR    LR, [SP, #-16]!
    LDP    Q2, Q3, [X0]
    FADD   V4.4S, V2.4S, V2.4S
    MOV    X4, X1
    ADD    X4, X4, #1
dp_while_loop:
    SUBS   X4, X4, #1
    B.EQ   dp_while_done
    FMUL   V1.4S, V2.4S, V3.4S
    FADDP  V0.4S, V1.4S, V1.4S
    FADDP  V0.4S, V0.4S, V0.4S
    FADD   V2.4S, V2.4S, V4.4S
    B      dp_while_loop
dp_while_done:
    FMOV   S0, S2
    LDR    LR, [SP], #16
    RET

```

Accelerate

```
func runAccTest(theA: Float, theB: Float, theReps: Int) -> Float {
    var tempA: Float = theA
    var vA = simd_float4(theA, theA, theA, theA)
    let vB = simd_float4(theB, theB, theB, theB)
    let vC = vA + vA
    for _ in 1...theReps {
        tempA += simd_dot(vA, vB)
        vA = vA + vC
    }
    return tempA
}
```

Mixed

```
func runSwiftTest(theA: Float, theB: Float, theReps: Int) -> Float {
    var tempA: Float = theA
    var vA = [theA, theA, theA, theA]
    let vB = [theB, theB, theB, theB]
    let vC = vA.map { $0 * 2.0 }
    for _ in 1...theReps {
        tempA = zip(vA, vB).map(*).reduce(0, +)
        for (index, value) in vA.enumerated() {
            vA[index] = value + vC[index]
        }
    }
    return tempA
}
```

Background processing etc.

This is performed using an OperationQueue with the set QoS.

maxConcurrentOperationCount is to the number of processes to be performed, with results being returned on the main OperationQueue. Times are obtained using mach_absolute_time().

Have fun!