Performance evaluation of AES (Advanced Encryption Standard) and AVX (Advanced Vector Extensions) hardware accelerators.

Subject: EECE7352 – Computer Architecture

By:

Likitha Muralidhar

Nayana Mahesh Patel

Content:

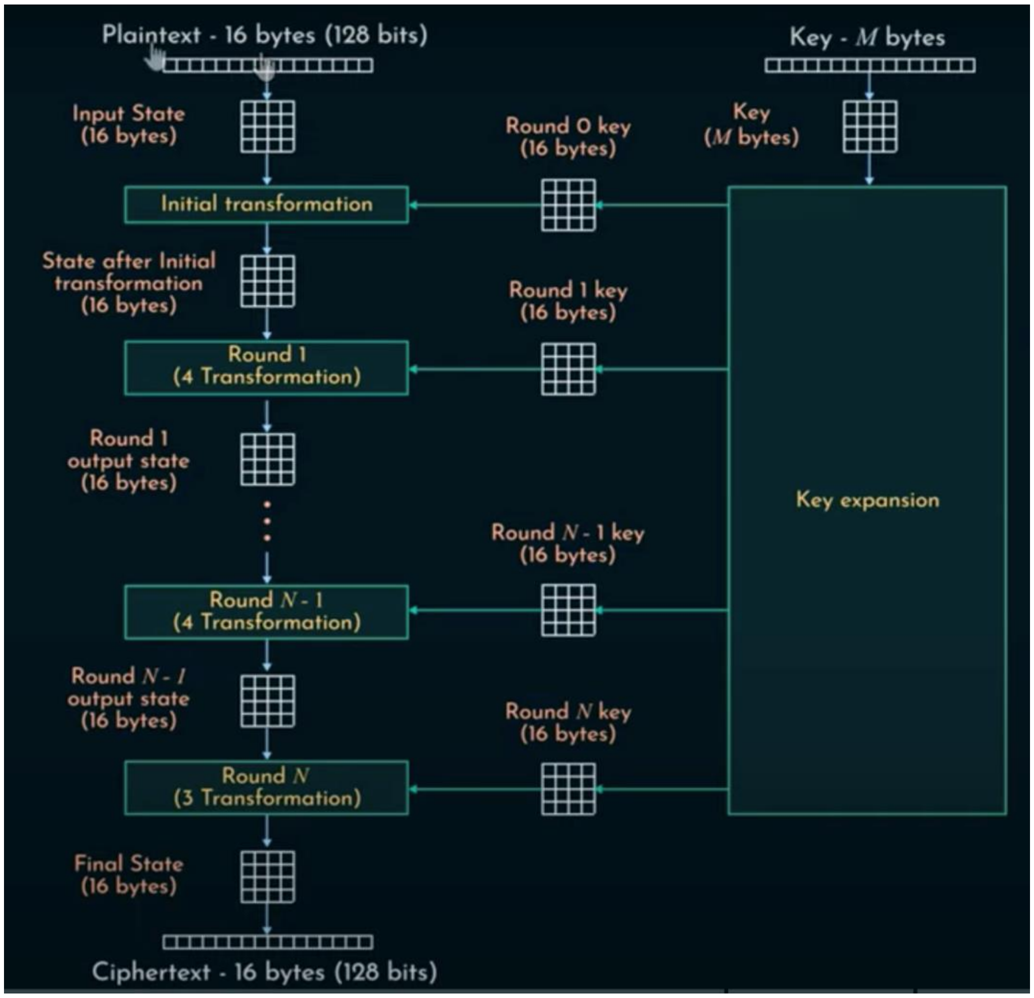
1.ABSTRACT:

Hardware accelerators are specialized hardware components that are connected to the CPU to improve the performance by offloading specific tasks. In today's general purpose PCs different hardware accelerators are supported to offload AI, cryptographic tasks, Graphics operations. We will look at two hardware accelerators and benchmark them on different platforms. AES is hardware accelerator for cryptographic operations such encryption and decryption. AVX is used to enhance the performance of vector processing workloads. The simulators used to assess the performance are OpenSSL for AES and Gem5 for AVX on two different CPU cores. The results provide insights into the design and utilization of hardware accelerators for secure and high-performance computing, emphasizing their critical role in contemporary processor architectures.

2.Advanced Encryption Standard (AES):

Advanced Encryption Standard is a symmetric block cipher algorithm that encrypts data for security purposes. AES is implemented in software and hardware to encrypt sensitive data. AES runs multiple encryption rounds and splits a message into smaller data blocks. AES was developed by the National Institute of Standards and Technology (NIST) beginning in 1997. In June 2003, AES was designated the default encryption technique for safeguarding classified information, including government secrets.

AES uses a key (cipher key) whose length can be 128, 192, or 256 bits. Hereafter encryption/decryption with a cipher key of 128, 192, or 256 bits is denoted AES-128, AES-192, and AES-256, respectively. AES-128, AES-192, and AES-256 process the data block in, respectively, 10, 12, or 14 iterations of pre-defined sequences of transformations, which are also called AES rounds ("rounds" for short). The rounds are identical except for the last one, which slightly differs from the others (by skipping one of the transformations). Each (N-1) round consists of 4 transformations namely substitute bytes, shift rows, mix columns, and addition of round key while the last stage only consists of three stages column mixing is not performed in the last round.


Plaintext - 16 bytes (128 bits)
Key - M bytes
Input State (16 bytes)
Round 0 key (16 bytes)
Key (M bytes)
Initial transformation
State after Initial transformation (16 bytes)
Round 1 key (16 bytes)
Round 1 (4 Transformation)
Round 1 output state (16 bytes)
Key expansion
Round N - 1 key (16 bytes)
Round N - 1 (4 Transformation)
Round N - 1 output state (16 bytes)
Round N key (16 bytes)
Round N (3 Transformation)
Final State (16 bytes)
Ciphertext - 16 bytes (128 bits)

The rounds operate on two 128-bit inputs: "State" and "Round key". Each round from 1 to 10/12/14 uses a different round key. The 10/12/14 round keys are derived from the cipher key by the "Key Expansion" algorithm.

This algorithm is independent of the processed data and can be therefore carried out independently of the encryption/decryption phase (typically, the key is expanded once and is thereafter used for many data blocks using some cipher mode of operation).

AES support in x86 architecture:

Intel introduced new set of instructions supporting AES from 2010 Intel core processor family based on the 32nm Intel microarchitecture codename Westmere.

Intel introduced 6 instructions supporting AES, four instructions (AESENC, AESENCLAST, AESDEC, and AESDELAST) facilitate high performance AES encryption and decryption, and the other two (AESIMC and AESKEYGENASSIST) support the AES key expansion.

2.1 Tool to benchmark AES modules:

OpenSSL:

The Open Secure Sockets Layer (OpenSSL) software library is a robust, commercial-grade, full-featured toolkit for general-purpose cryptography and secure communication.

The OpenSSL program provides a rich variety of commands, each of which often has a wealth of options and arguments. OpenSSL provides command line support to benchmark cryptographic algorithms.

- 'openssl speed' command is used for benchmarking different cryptographic algorithms.[6]
- openssl speed [-help] [-engine id] [-elapsed] [-evp algo] [-decrypt] [-rand file...] [-writerand file] [-primes num] [-seconds num] [-bytes num] [algorithm...]

Example command: openssl speed -evp aes-128-cbc

- Here use of -evp flag ensures that it uses hardware accelerator that is present in the system and without this flag it will not run these cryptography operations on specialized hardware.
- 128 represents key size
- cbc represents mode of AES encryption

Supported key sizes in bits: 128,192,256

Supported modes:

- ECB mode: Electronic Code Book
- CBC mode: Cipher Block Chaining
- CFB mode: Cipher Feed Back
- OFB mode: Output Feed Back
- CTR mode: Counter

2.2 Results:

Intel-COE

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                16
On-line CPU(s) list:   0-15
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Model name:            Intel(R) Xeon(R) CPU         E5620  @ 2.40GHz
Stepping:              2
CPU MHz:               1600.000
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx
fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid dca sse4_1
sse4_2 popcnt aes lahf_lm epb tpr_shadow vnmi flexpriority ept vpid dtherm ida arat
```

Performance with -evp flag used:

Key-size	Block size	ECB	CBC	CFB	OFB	CTR
128-bit	16 bytes	576485.90k	569798.20k	418553.82k	441019.35k	400023.60k
	64 bytes	1548092.29k	675677.97k	460222.40k	571182.61k	995774.44k
	256 bytes	1897217.79k	692192.34k	462056.70k	598559.57k	1645119.91k
192-bit	16 bytes	491314.31k	484984.24k	370476.85k	378665.35k	385096.57k
	64 bytes	1351269.23k	565811.99k	419410.82k	486506.28k	921821.50k
	256 bytes	1606341.97k	578784.43k	424109.57k	510099.20k	1424643.33k
256-bit	16 bytes	427095.71k	422225.34k	333012.01k	338427.18k	339059.39k
	64 bytes	1207738.37k	489341.85k	374952.09k	427929.37k	851117.91k
	256 bytes	1387054.68k	497956.44k	378701.82k	446292.99k	1253901.31k

Note: 1000s of bytes per second processed.

Performance without -evp flag used:

Key-size	Block size	CBC
128-bit	16 bytes	73700.44k
	64 bytes	86203.88k
	256 bytes	88208.73k
192-bit	16 bytes	62322.04k
	64 bytes	72102.44k
	256 bytes	73246.81k
256-bit	16 bytes	53850.23k
	64 bytes	61924.95k
	256 bytes	60526.51k

From the results taken in table it is clearly visible that with -evp flag (which enables use of AES hardware capability), it is operating on a greater number of bytes compared to without AES capabilities.

AES support in ARM:

ARM cryptographic extensions are optionally supported on ARM Cortex-A30/50/70 cores. It supports five instructions for AES operations, AESD, AESE, AESIMC, AESMC, VMULL

ARM – COE

Architecture:	aarch64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	6
On-line CPU(s) list:	0,3-5
Off-line CPU(s) list:	1,2
Thread(s) per core:	1
Core(s) per socket:	4
Socket(s):	1
Vendor ID:	ARM
Model:	3
Model name:	Cortex-A57
Stepping:	r1p3
CPU max MHz:	2035.2000
Flags:	fp asimd evtstrm aes pmull sha1 sha2 crc32

Performance with -evp flag used:

Key-size	**Block size**	**ECB**	**CBC**	**CFB**	**OFB**	**CTR**
128-bit	16 bytes	430868.18k	371391.55k	258728.59k	271816.35k	241549.69k
	64 bytes	703679.62k	719069.23k	456865.22k	496328.75k	720918.89k
	256 bytes	997373.87k	898285.06k	576626.18k	659057.66k	1367567.27k
192-bit	16 bytes	397112.51k	300433.40k	249315.42k	226693.52k	231261.11k
	64 bytes	634661.55k	572969.19k	425926.34k	427273.90k	676144.94k
	256 bytes	834505.98k	746839.47k	536145.92k	596386.13k	1241740.80k
256-bit	16 bytes	371559.48k	326345.57k	239527.47k	252138.39k	219841.59k
	64 bytes	581346.99k	559516.12k	393961.47k	439001.30k	633566.87k
	256 bytes	764484.27k	666128.04k	474253.65k	571337.90k	1135822.25k

Note: 1000s of bytes per second processed.

Performance without -evp flag used:

Key-size	**Block size**	**CBC**
128-bit	16 bytes	94125.02k
	64 bytes	98970.65k
	256 bytes	101928.45k
192-bit	16 bytes	76403.48k
	64 bytes	83119.89k
	256 bytes	86488.15k
256-bit	16 bytes	71503.93k
	64 bytes	74291.14k
	256 bytes	75988.31k

Performance of AES algorithm depends on the processor frequency, memory subsystem (mainly organization of cache), and number of threads supported.

3. AVX (Advanced Vector Extensions)

Advanced Vector Extension (AVX) are SIMD extensions to the x86 instruction set architecture for microprocessors from Intel and AMD. These x86 AVX accelerator is a set of instructions that can boost performance for vector processing–intensive workloads. Vector processing, an essential part of many advanced computational tasks, performs an arithmetic operation on a large array of integers or floating-point numbers in parallel.

Different Versions of AVX

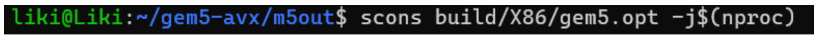
1. AVX – For 128 bits (Uses XMMx registers)
2. AVX2 – for 256 bits (Uses YMMx registers)
3. AVX-512 – for 521 bits (Uses ZMMx registers)

3.1 Simulator Used to Evaluate AVX performance

The gem5 simulator is a modular platform for computer-system architecture research, encompassing system-level architecture as well as processor microarchitecture.

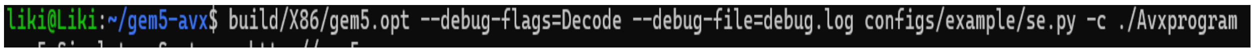
Gem5 supports AVX extension, and by using the gem5 we have evaluated the AVX accelerators performance on two different CPU cores by taking a simple AVX workload

Command to build Gem5:


```
liki@Liki:~/gem5-avx/m5out$ scons build/X86/gem5.opt -j$(nproc)
```

After gem5 is successfully built

Command to run the AVX workload with decode instructions:


```
liki@Liki:~/gem5-avx$ build/X86/gem5.opt --debug-flags=Decode --debug-file=debug.log configs/example/se.py -c ./Avxprogram
```

3.2 AVX workload Used:

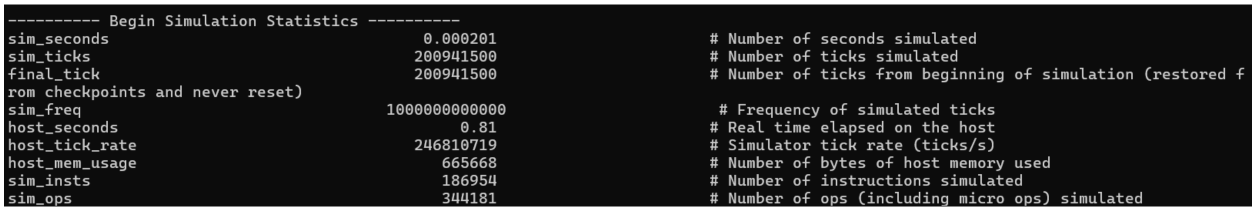
```
#include <immintrin.h>
#include <stdio.h>

int main() {
   float a[8] __attribute__((aligned(32))) = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0};
   float b[8] __attribute__((aligned(32))) = {8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0};
   float result[8] __attribute__((aligned(32)));

   // Load vectors into AVX registers
   __m256 vec_a = _mm256_load_ps(a);
   __m256 vec_b = _mm256_load_ps(b);

   // Perform AVX addition
   __m256 vec_result = _mm256_add_ps(vec_a, vec_b);

   // Store the result
   _mm256_store_ps(result, vec_result);

   // Print the result
   for (int i = 0; i < 8; ++i) {
      printf("result[%d] = %f\n", i, result[i]);
   }
   return 0;
}
```

Experiment was done on two different CPU cores, these cores details and experiment outputs are given below

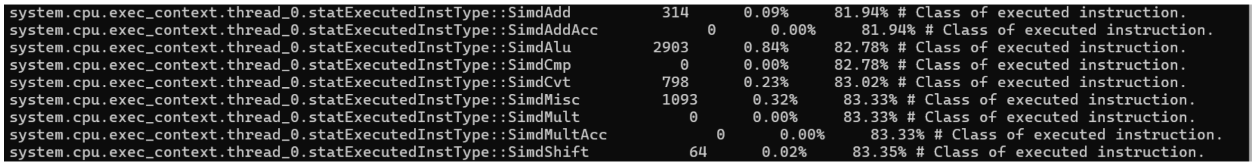
CPU1 Details:

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Address sizes:	46 bits physical, 48 bits virtual
Byte Order:	Little Endian
CPU(s):	14
On-line CPU(s) list:	0-13
Vendor ID:	Genuine Intel
Model name:	Intel(R) Core(TM) Ultra 5 125U
CPU family:	6
Model:	170
Thread(s) per core:	2
Core(s) per socket:	7
Socket(s):	1
Stepping:	4
BogoMIPS:	5376.00
Flags:	fpu vme de pse tsc msr pae mce cx8 apic sep mtrr p ge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_ good nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4 _2 x2apic movbe popcnt tsc_deadline_timer aes xsav e avx f16c rdrand hypervisor lahf_lm abm 3dnowpref etch invpcid_single ssbd ibrs ibpb stibp ibrs_enha nced tpr_shadow vnmi ept vpid ept_ad fsgsbase tsc_ adjust bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetb v1 xsaves avx_vnni umip waitpkg gfni vaes vpclmulq dq rdpid movdiri movdir64b fsrm md_clear serialize flush_l1d arch_capabilities

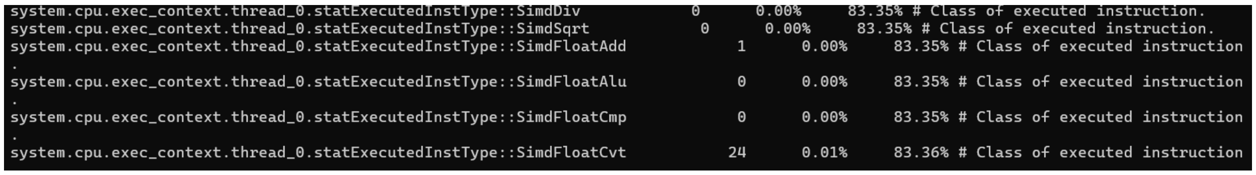
Output: (Output will be generated in the file name stats.txt)



```
---------- Begin Simulation Statistics ----------
sim_seconds                                  0.000201                       # Number of seconds simulated
sim_ticks                                   200941500                       # Number of ticks simulated
final_tick                                  200941500                       # Number of ticks from beginning of simulation (restored f
rom checkpoints and never reset)
sim_freq                                1000000000000                        # Frequency of simulated ticks
host_seconds                                     0.81                       # Real time elapsed on the host
host_tick_rate                              246810719                       # Simulator tick rate (ticks/s)
host_mem_usage                                 665668                       # Number of bytes of host memory used
sim_insts                                      186954                       # Number of instructions simulated
sim_ops                                        344181                       # Number of ops (including micro ops) simulated
```

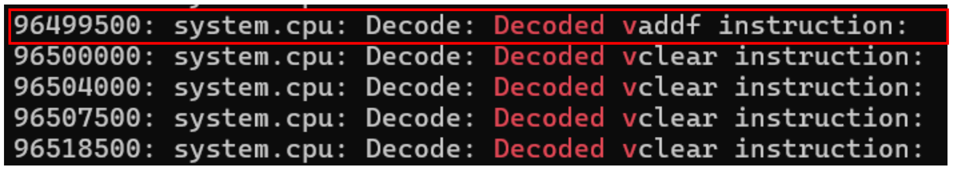


```
system.cpu.exec_context.thread_0.statExecutedInstType::SimdAdd          314      0.09%     81.94% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdAddAcc            0      0.00%     81.94% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdAlu         2903      0.84%     82.78% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdCmp            0      0.00%     82.78% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdCvt          798      0.23%     83.02% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdMisc          1093      0.32%     83.33% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdMult            0      0.00%     83.33% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdMultAcc            0      0.00%     83.33% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdShift           64      0.02%     83.35% # Class of executed instruction.
```



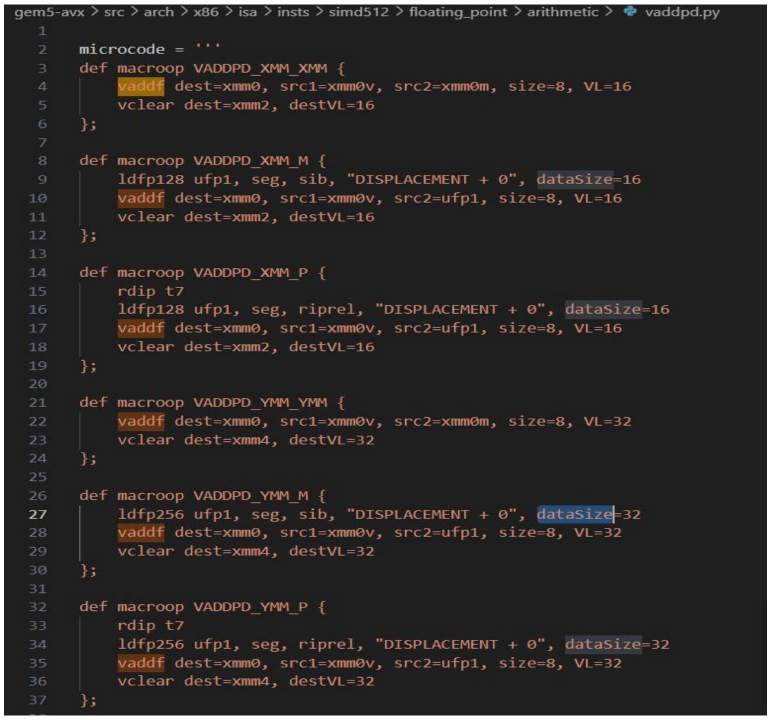
```
system.cpu.exec_context.thread_0.statExecutedInstType::SimdDiv            0      0.00%     83.35% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdSqrt            0      0.00%     83.35% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatAdd            1      0.00%     83.35% # Class of executed instruction
.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatAlu            0      0.00%     83.35% # Class of executed instruction
.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatCmp            0      0.00%     83.35% # Class of executed instruction
.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatCvt           24      0.01%     83.36% # Class of executed instruction
```

When decoded the instruction used for AVX work



```
96499500: system.cpu: Decode: Decoded vaddf instruction:
96500000: system.cpu: Decode: Decoded vclear instruction:
96504000: system.cpu: Decode: Decoded vclear instruction:
96507500: system.cpu: Decode: Decoded vclear instruction:
96518500: system.cpu: Decode: Decoded vclear instruction:
```

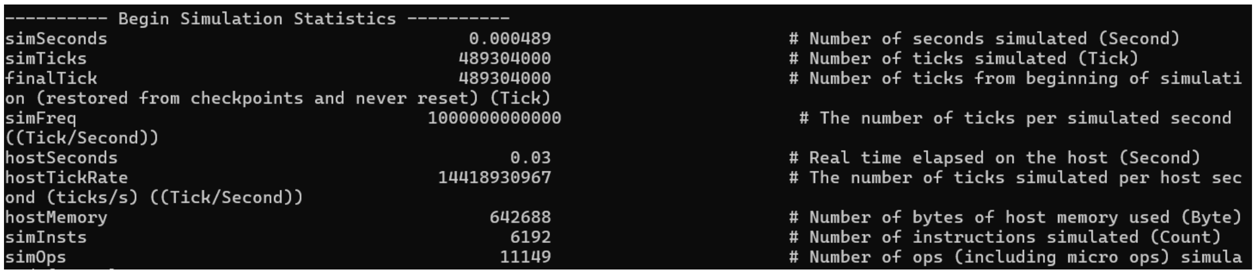
Note: In gem5 it configured such that whatever the vector additions instruction it uses either Vaddpd or vaddps it will take as vaddf.



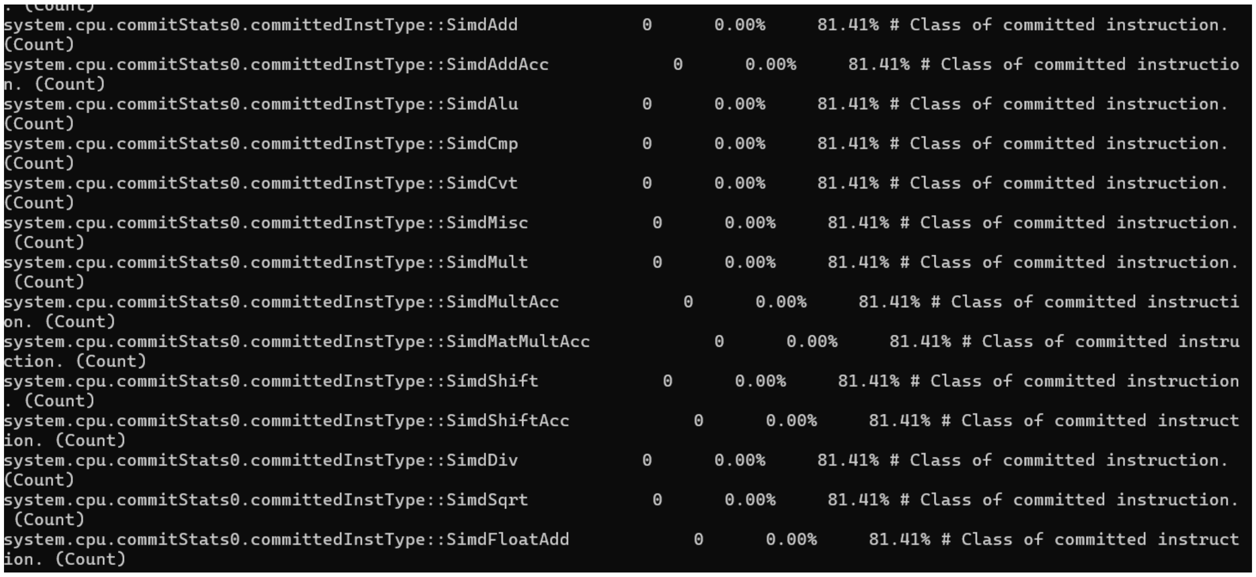
gem5-avx > src > arch > x86 > isa > insts > simd512 > floating_point > arithmetic > vaddpd.py

```

microcode = '''
def macroop VADDPD_XMM_XMM {
    vaddf dest=xmm0, src1=xmm0v, src2=xmm0m, size=8, VL=16
    vclear dest=xmm2, destVL=16
};

def macroop VADDPD_XMM_M {
    ldfp128 ufp1, seg, sib, "DISPLACEMENT + 0", dataSize=16
    vaddf dest=xmm0, src1=xmm0v, src2=ufp1, size=8, VL=16
    vclear dest=xmm2, destVL=16
};

def macroop VADDPD_XMM_P {
    rdip t7
    ldfp128 ufp1, seg, riprel, "DISPLACEMENT + 0", dataSize=16
    vaddf dest=xmm0, src1=xmm0v, src2=ufp1, size=8, VL=16
    vclear dest=xmm2, destVL=16
};

def macroop VADDPD_YMM_YMM {
    vaddf dest=xmm0, src1=xmm0v, src2=xmm0m, size=8, VL=32
    vclear dest=xmm4, destVL=32
};

def macroop VADDPD_YMM_M {
    ldfp256 ufp1, seg, sib, "DISPLACEMENT + 0", dataSize=32
    vaddf dest=xmm0, src1=xmm0v, src2=ufp1, size=8, VL=32
    vclear dest=xmm4, destVL=32
};

def macroop VADDPD_YMM_P {
    rdip t7
    ldfp256 ufp1, seg, riprel, "DISPLACEMENT + 0", dataSize=32
    vaddf dest=xmm0, src1=xmm0v, src2=ufp1, size=8, VL=32
    vclear dest=xmm4, destVL=32
};
```

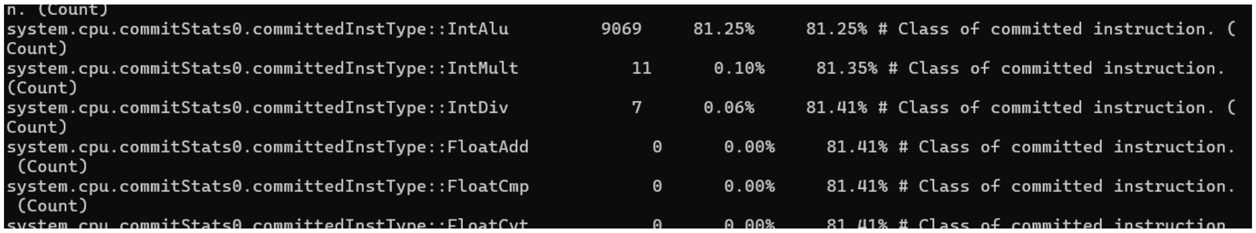
Output on the same CPU without using AVX instructions:



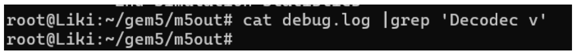
```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.000489                       # Number of seconds simulated (Second)
simTicks                                    489304000                       # Number of ticks simulated (Tick)
finalTick                                   489304000                       # Number of ticks from beginning of simulati
on (restored from checkpoints and never reset) (Tick)
simFreq                                  1000000000000                       # The number of ticks per simulated second
((Tick/Second))
hostSeconds                                      0.03                       # Real time elapsed on the host (Second)
hostTickRate                              14418930967                       # The number of ticks simulated per host sec
ond (ticks/s) ((Tick/Second))
hostMemory                                     642688                       # Number of bytes of host memory used (Byte)
simInsts                                         6192                       # Number of instructions simulated (Count)
simOps                                          11149                       # Number of ops (including micro ops) simula
```



```
. (Count)
system.cpu.commitStats0.committedInstType::SimdAdd            0      0.00%     81.41% # Class of committed instruction.
(Count)
system.cpu.commitStats0.committedInstType::SimdAddAcc            0      0.00%     81.41% # Class of committed instructio
n. (Count)
system.cpu.commitStats0.committedInstType::SimdAlu            0      0.00%     81.41% # Class of committed instruction.
(Count)
system.cpu.commitStats0.committedInstType::SimdCmp            0      0.00%     81.41% # Class of committed instruction.
(Count)
system.cpu.commitStats0.committedInstType::SimdCvt            0      0.00%     81.41% # Class of committed instruction.
(Count)
system.cpu.commitStats0.committedInstType::SimdMisc            0      0.00%     81.41% # Class of committed instruction.
 (Count)
system.cpu.commitStats0.committedInstType::SimdMult            0      0.00%     81.41% # Class of committed instruction.
 (Count)
system.cpu.commitStats0.committedInstType::SimdMultAcc            0      0.00%     81.41% # Class of committed instructi
on. (Count)
system.cpu.commitStats0.committedInstType::SimdMatMultAcc            0      0.00%     81.41% # Class of committed instru
ction. (Count)
system.cpu.commitStats0.committedInstType::SimdShift            0      0.00%     81.41% # Class of committed instruction
. (Count)
system.cpu.commitStats0.committedInstType::SimdShiftAcc            0      0.00%     81.41% # Class of committed instruct
ion. (Count)
system.cpu.commitStats0.committedInstType::SimdDiv            0      0.00%     81.41% # Class of committed instruction.
(Count)
system.cpu.commitStats0.committedInstType::SimdSqrt            0      0.00%     81.41% # Class of committed instruction.
 (Count)
system.cpu.commitStats0.committedInstType::SimdFloatAdd            0      0.00%     81.41% # Class of committed instruct
ion. (Count)
```



```
n. (Count)
system.cpu.commitStats0.committedInstType::IntAlu         9069     81.25%     81.25% # Class of committed instruction. (
Count)
system.cpu.commitStats0.committedInstType::IntMult           11      0.10%     81.35% # Class of committed instruction.
(Count)
system.cpu.commitStats0.committedInstType::IntDiv           7      0.06%     81.41% # Class of committed instruction. (
Count)
system.cpu.commitStats0.committedInstType::FloatAdd            0      0.00%     81.41% # Class of committed instruction.
 (Count)
system.cpu.commitStats0.committedInstType::FloatCmp            0      0.00%     81.41% # Class of committed instruction.
 (Count)
system.cpu.commitStats0.committedInstType::FloatCvt            0      0.00%     81.41% # Class of committed instruction.
```

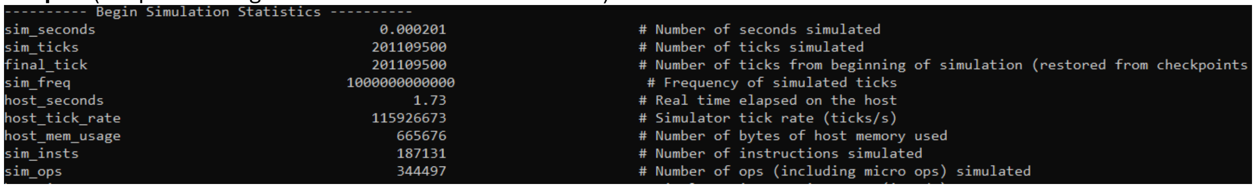


```
root@Liki:~/gem5/m5out# cat debug.log |grep 'Decodec v'
root@Liki:~/gem5/m5out#
```

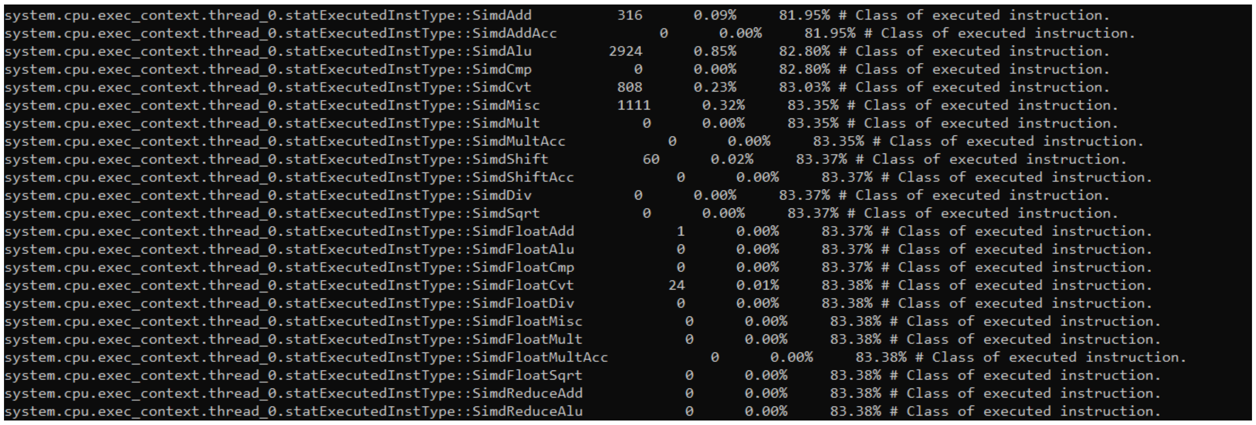
CPU2 details:

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
Address sizes:	39 bits physical, 48 bits virtual
CPU(s):	8
On-line CPU(s) list:	0-7
Thread(s) per core:	2
Core(s) per socket:	4
Socket(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	140
Model name:	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
Stepping:	1
CPU MHz:	2803.212
BogoMIPS:	5606.42
Hypervisor vendor:	Microsoft
Virtualization type:	full
L1d cache:	192 KiB
L1i cache:	128 KiB
L2 cache:	5 MiB
L3 cache:	12 MiB
Flags:	fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq ssse3 fma cx16 pdcm pcid sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced fsgsbase bmi1 avx2 smep bmi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt clwb avx512cd sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves avx512vbmi umip avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx512_bitalg avx512_vpopcntdq rdpid fsrm avx512_vp2intersect md_clear flush_l1d arch_capabilities

Output: (Output will be generated in the file name stats.txt)

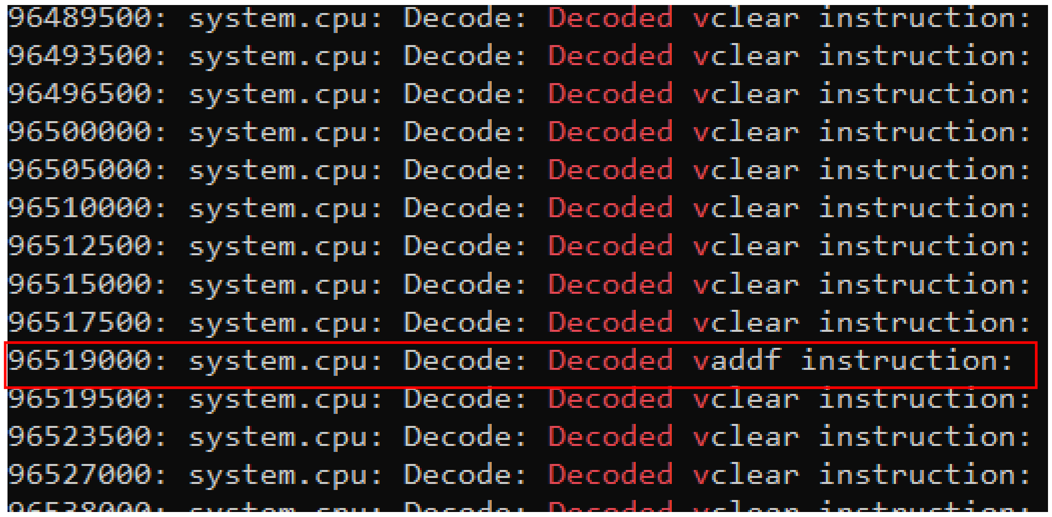


```
---------- Begin Simulation Statistics ----------
sim_seconds                                  0.000201                       # Number of seconds simulated
sim_ticks                                   201109500                       # Number of ticks simulated
final_tick                                  201109500                       # Number of ticks from beginning of simulation (restored from checkpoints
sim_freq                                 1000000000000                       # Frequency of simulated ticks
host_seconds                                     1.73                       # Real time elapsed on the host
host_tick_rate                              115926673                       # Simulator tick rate (ticks/s)
host_mem_usage                                 665676                       # Number of bytes of host memory used
sim_insts                                      187131                       # Number of instructions simulated
sim_ops                                        344497                       # Number of ops (including micro ops) simulated
```



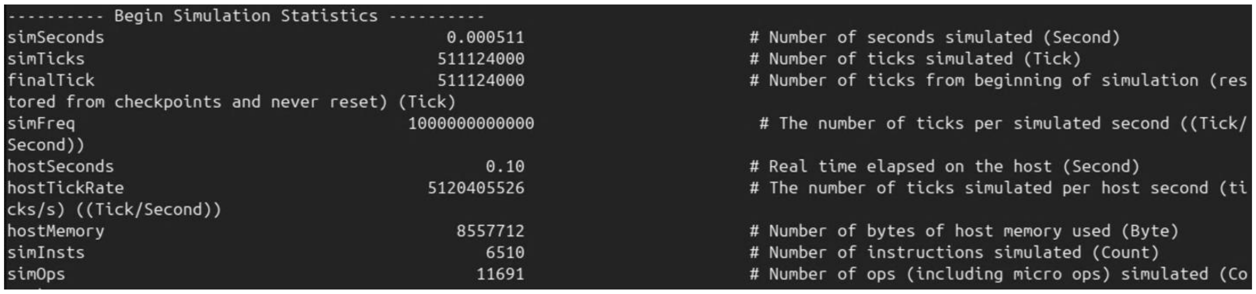
```
system.cpu.exec_context.thread_0.statExecutedInstType::SimdAdd          316      0.09%     81.95% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdAddAcc          0      0.00%     81.95% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdAlu         2924      0.85%     82.80% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdCmp            0      0.00%     82.80% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdCvt          808      0.23%     83.03% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdMisc         1111      0.32%     83.35% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdMult            0      0.00%     83.35% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdMultAcc            0      0.00%     83.35% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdShift           60      0.02%     83.37% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdShiftAcc            0      0.00%     83.37% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdDiv            0      0.00%     83.37% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdSqrt            0      0.00%     83.37% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatAdd            1      0.00%     83.37% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatAlu            0      0.00%     83.37% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatCmp            0      0.00%     83.37% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatCvt           24      0.01%     83.38% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatDiv            0      0.00%     83.38% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatMisc            0      0.00%     83.38% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatMult            0      0.00%     83.38% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatMultAcc            0      0.00%     83.38% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdFloatSqrt            0      0.00%     83.38% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdReduceAdd            0      0.00%     83.38% # Class of executed instruction.
system.cpu.exec_context.thread_0.statExecutedInstType::SimdReduceAlu            0      0.00%     83.38% # Class of executed instruction.
```

When decoded the instruction used for AVX work

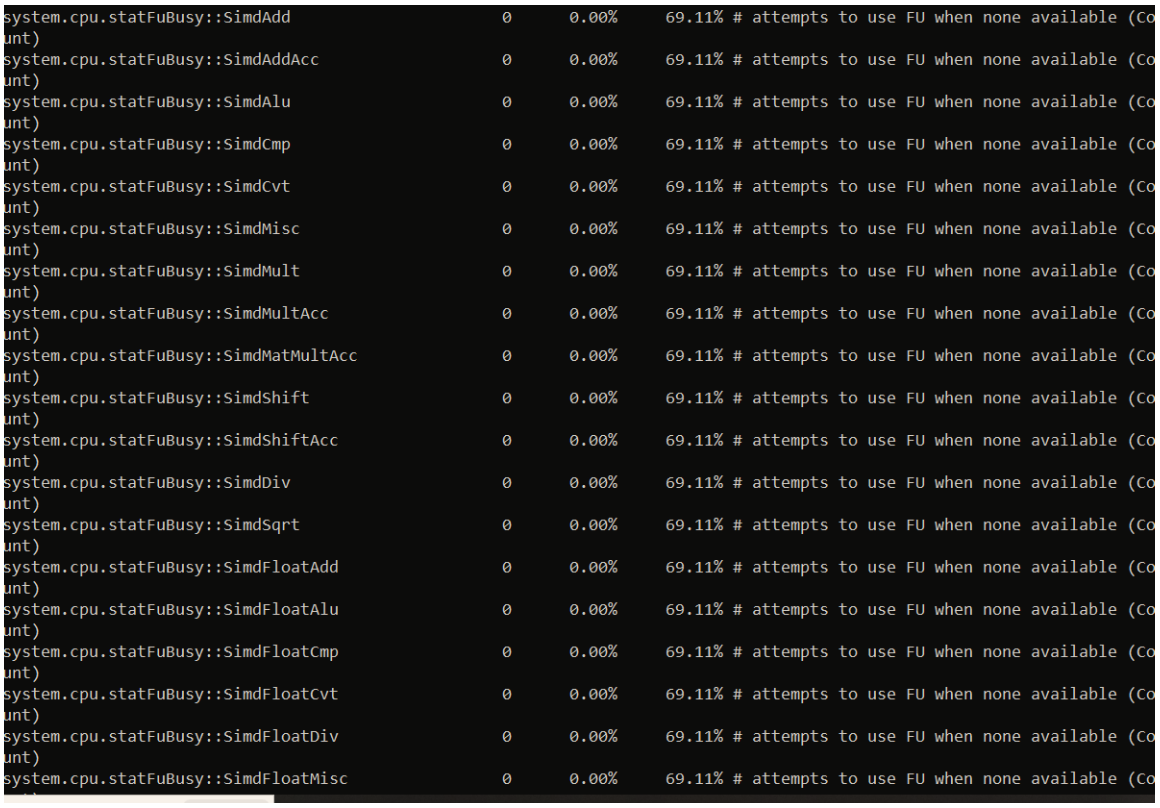


```
96489500: system.cpu: Decode: Decoded vclear instruction:
96493500: system.cpu: Decode: Decoded vclear instruction:
96496500: system.cpu: Decode: Decoded vclear instruction:
96500000: system.cpu: Decode: Decoded vclear instruction:
96505000: system.cpu: Decode: Decoded vclear instruction:
96510000: system.cpu: Decode: Decoded vclear instruction:
96512500: system.cpu: Decode: Decoded vclear instruction:
96515000: system.cpu: Decode: Decoded vclear instruction:
96517500: system.cpu: Decode: Decoded vclear instruction:
96519000: system.cpu: Decode: Decoded vaddf instruction:
96519500: system.cpu: Decode: Decoded vclear instruction:
96523500: system.cpu: Decode: Decoded vclear instruction:
96527000: system.cpu: Decode: Decoded vclear instruction:
```

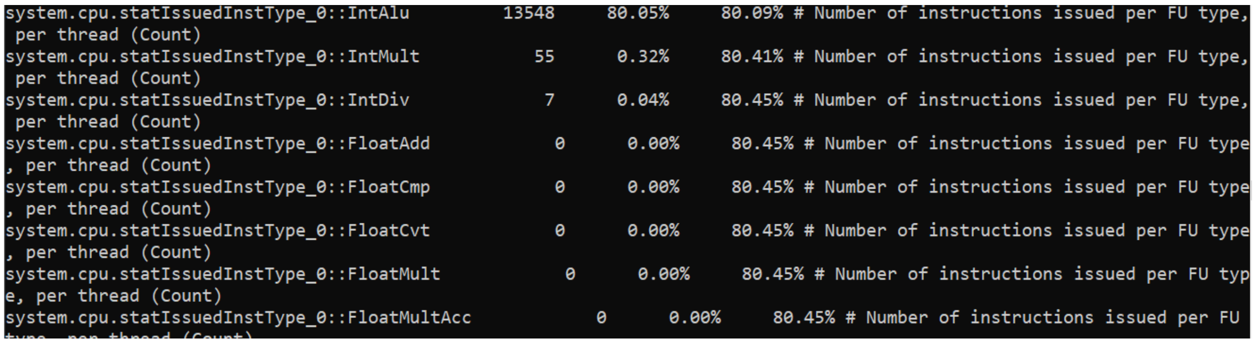
Output on the same CPU without using AVX instructions:



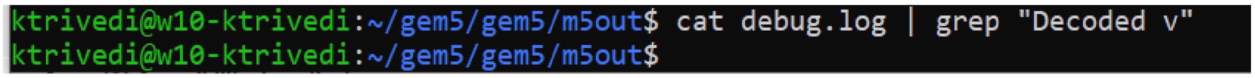
```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.000511                       # Number of seconds simulated (Second)
simTicks                                    511124000                       # Number of ticks simulated (Tick)
finalTick                                   511124000                       # Number of ticks from beginning of simulation (res
tored from checkpoints and never reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/
Second))
hostSeconds                                      0.10                       # Real time elapsed on the host (Second)
hostTickRate                               5120405526                       # The number of ticks simulated per host second (ti
cks/s) ((Tick/Second))
hostMemory                                    8557712                       # Number of bytes of host memory used (Byte)
simInsts                                         6510                       # Number of instructions simulated (Count)
simOps                                          11691                       # Number of ops (including micro ops) simulated (Co
```



```
system.cpu.statFuBusy::SimdAdd                   0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdAddAcc                0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdAlu                   0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdCmp                   0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdCvt                   0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdMisc                  0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdMult                  0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdMultAcc               0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdMatMultAcc            0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdShift                 0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdShiftAcc              0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdDiv                   0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdSqrt                  0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdFloatAdd              0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdFloatAlu              0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdFloatCmp              0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdFloatCvt              0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdFloatDiv              0      0.00%     69.11% # attempts to use FU when none available (Co
unt)
system.cpu.statFuBusy::SimdFloatMisc             0      0.00%     69.11% # attempts to use FU when none available (Co
```



```
system.cpu.statIssuedInstType_0::IntAlu         13548     80.05%     80.09% # Number of instructions issued per FU type,
 per thread (Count)
system.cpu.statIssuedInstType_0::IntMult           55      0.32%     80.41% # Number of instructions issued per FU type,
 per thread (Count)
system.cpu.statIssuedInstType_0::IntDiv             7      0.04%     80.45% # Number of instructions issued per FU type,
 per thread (Count)
system.cpu.statIssuedInstType_0::FloatAdd           0      0.00%     80.45% # Number of instructions issued per FU type
, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCmp           0      0.00%     80.45% # Number of instructions issued per FU type
, per thread (Count)
system.cpu.statIssuedInstType_0::FloatCvt           0      0.00%     80.45% # Number of instructions issued per FU type
, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMult           0      0.00%     80.45% # Number of instructions issued per FU typ
e, per thread (Count)
system.cpu.statIssuedInstType_0::FloatMultAcc           0      0.00%     80.45% # Number of instructions issued per FU
```



```
ktrivedi@w10-ktrivedi:~/gem5/gem5/m5out$ cat debug.log | grep "Decoded v"
ktrivedi@w10-ktrivedi:~/gem5/gem5/m5out$
```

3.3 Result:

Table1: Comparison of Experiment Results of both the CPU

	CPU1(Intel core Ultra5)		CPU2(Intel core i7)	
	With AVX	Without AVX	With AVX	Without AVX
Sim_seconds	0.000201	0.000489	0.000201	0.000511
Sim_ticks	200941500	489304000	201109500	511124000
SIMDadd(No of Instructions)	314	0	316	0
SIMDalu(No of Instructions)	2903	0	2924	0
SIMDCvt(No of Instructions)	789	0	808	0
SIMD Misc(No of Instructions)	1093	0	1111	0
SIMDShift(No of Instructions)	64	0	60	0
SIMDFloat add(No of Instructions)	1	0	1	0
SIMD Floatcvt(No of Instructions)	24	0	24	0

From the table it can be understood that on both the CPU cores, the simulation seconds for computing the workload is less in case of AVX instruction when compared to without AVX instruction. Also, there are no SIMD instructions when workload was computed without AVX instruction. So, with the usage of AVX instructions the computation time of vector processing workloads can be reduced.

5. References:

[1] https://en.wikipedia.org/wiki/Advanced_Vector_Extensions

[2] S. Lee, Y. Kim, D. Nam and J. Kim, "Gem5-AVX: Extension of the Gem5 Simulator to Support AVX Instruction Sets," in *IEEE Access*, vol. 12, pp. 20767-20778, 2024, doi: 10.1109/ACCESS.2024.3359296. keywords: {Instruction sets;Decoding;Computer architecture;Codes;Syntactics;Supercomputers;Memory management;Benchmark testing;Support vector machines;Program processors;Gem5 simulator;x86 SIMD;AVX;AVX2;AVX-512},

[3] https://github.com/seanzw/gem5-avx.git

[4] M. Pashinska-Gadzheva, "Comparison of compiler efficiency with SSE and AVX instructions," *2022 International Conference Automatics and Informatics (ICAI)*, Varna, Bulgaria, 2022, pp. 56-59, doi: 10.1109/ICAI55857.2022.9960080.

[5] Abdullah, Ako. (2017). Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data.

[6] R. Tomoiaga and M. Stratulat, "AES Performance Analysis on Several Programming Environments, Operating Systems or Computational Platforms," *2010 Fifth International Conference on Systems and Networks Communications*, Nice, France, 2010, pp. 172-176, doi: 10.1109/ICSNC.2010.33.

[7] Intel® Intrinsics Guide