Vectorization

Write-up 1: Look at the assembly code above. The compiler has translated the code to set the start index at -2^{16} and adds to it for each memory access. Why doesn't it set the start index to 0 and use small positive offsets?

答：每一次循环，i都需要和SIZE（即2的16次方）作比较。先将内存置为-2的16次方，加上i后直接判断正负来比较大小，可以减少开销。

Write-up 2: This code is still not aligned when using AVX2 registers. Fix the code to make sure it uses aligned moves for the best performance.

答：修改代码如下，使程序使用32位对齐寄存器性能最优

```
#include <stdint.h>
#include <stdlib.h>
#include <math.h>

#define SIZE (1L << 32)

void test(uint8_t* restrict a, uint8_t* restrict b) {
  a = __builtin_assume_aligned(a, 32);
  b = __builtin_assume_aligned(b, 32);

  for (uint64_t i = 0; i < SIZE; i++) {
    a[i] += b[i];
  }
}
```

Write-up 3: Provide a theory for why the compiler is generating dramatically different assembly.

答：三元运算符不会按元素比较数组。

Write-up 4: Inspect the assembly and determine why the assembly does not include instructions with vector registers. Do you think it would be faster if it did vectorize? Explain.

答:

运行生成的汇编代码如下:

```
	.text
	.file	"example3.c"
	.globl	test                    # -- Begin function test
	.p2align	4, 0x90
	.type	test,@function
test:                                   # @test
.Lfunc_begin0:
	.file	1 "/home/zhangkeer/a4/recitation3" "example3.c"
	.loc	1 9 0                   # example3.c:9:0
	.cfi_startproc
# %bb.0:
	#DEBUG_VALUE: test:a <- $rdi
	#DEBUG_VALUE: test:a <- $rdi
	#DEBUG_VALUE: test:b <- $rsi
	#DEBUG_VALUE: test:b <- $rsi
	pushq	%rax
	.cfi_def_cfa_offset 16
.Ltmp0:
	#DEBUG_VALUE: test:i <- 0
	.loc	1 12 3 prologue_end     # example3.c:12:3
	addq	$1, %rsi
.Ltmp1:
	.loc	1 13 10                 # example3.c:13:10
	movl	$65536, %edx            # imm = 0x10000
	#DEBUG_VALUE: test:a <- $rdi
	callq	memcpy
.Ltmp2:
	#DEBUG_VALUE: test:i <- undef
	.loc	1 15 1                  # example3.c:15:1
	popq	%rax
	.cfi_def_cfa_offset 8
	retq
.Ltmp3:
.Lfunc_end0:
	.size	test, .Lfunc_end0-test
	.cfi_endproc
                                        # -- End function
	.file	2 "/usr/include/x86_64-linux-gnu/bits" "types.h"
	.file	3 "/usr/include/x86_64-linux-gnu/bits" "stdint-uintn.h"
	.section	.debug_str,"MS",@progbits,1
.Linfo_string0:
	.asciz	"clang version 10.0.0-4ubuntu1 " # string offset=0
.Linfo_string1:
	.asciz	"example3.c"            # string offset=31
.Linfo_string2:
	.asciz	"/home/zhangkeer/a4/recitation3" # string offset=42
.Linfo_string3:
	.asciz	"test"                  # string offset=73
.Linfo_string4:
	.asciz	"a"                     # string offset=78
.Linfo_string5:
	.asciz	"unsigned char"         # string offset=80
```

```
.Linfo_string6:
    .asciz  "__uint8_t"             # string offset=94
.Linfo_string7:
    .asciz  "uint8_t"               # string offset=104
.Linfo_string8:
    .asciz  "b"                     # string offset=112
.Linfo_string9:
    .asciz  "i"                     # string offset=114
.Linfo_string10:
    .asciz  "long unsigned int"     # string offset=116
.Linfo_string11:
    .asciz  "__uint64_t"            # string offset=134
.Linfo_string12:
    .asciz  "uint64_t"              # string offset=145
    .section    .debug_loc,"",@progbits
.Ldebug_loc0:
    .quad   .Lfunc_begin0-.Lfunc_begin0
    .quad   .Ltmp2-.Lfunc_begin0
    .short  1                       # Loc expr size
    .byte   85                      # DW_OP_reg5
    .quad   0
    .quad   0
.Ldebug_loc1:
    .quad   .Lfunc_begin0-.Lfunc_begin0
    .quad   .Ltmp1-.Lfunc_begin0
    .short  1                       # Loc expr size
    .byte   84                      # DW_OP_reg4
    .quad   0
    .quad   0
.Ldebug_loc2:
    .quad   .Ltmp0-.Lfunc_begin0
    .quad   .Ltmp2-.Lfunc_begin0
    .short  2                       # Loc expr size
    .byte   48                      # DW_OP_lit0
    .byte   159                     # DW_OP_stack_value
    .quad   0
    .quad   0
    .section    .debug_abbrev,"",@progbits
    .byte   1                       # Abbreviation Code
    .byte   17                      # DW_TAG_compile_unit
    .byte   1                       # DW_CHILDREN_yes
    .byte   37                      # DW_AT_producer
    .byte   14                      # DW_FORM_strp
    .byte   19                      # DW_AT_language
    .byte   5                       # DW_FORM_data2
    .byte   3                       # DW_AT_name
    .byte   14                      # DW_FORM_strp
    .byte   16                      # DW_AT_stmt_list
    .byte   23                      # DW_FORM_sec_offset
    .byte   27                      # DW_AT_comp_dir
    .byte   14                      # DW_FORM_strp
    .byte   17                      # DW_AT_low_pc
    .byte   1                       # DW_FORM_addr
    .byte   18                      # DW_AT_high_pc
    .byte   6                       # DW_FORM_data4
    .byte   0                       # EOM(1)
    .byte   0                       # EOM(2)
    .byte   2                       # Abbreviation Code
```

```
	.byte	46                      # DW_TAG_subprogram
	.byte	1                       # DW_CHILDREN_yes
	.byte	17                      # DW_AT_low_pc
	.byte	1                       # DW_FORM_addr
	.byte	18                      # DW_AT_high_pc
	.byte	6                       # DW_FORM_data4
	.byte	64                      # DW_AT_frame_base
	.byte	24                      # DW_FORM_exprloc
	.ascii	"\227B"                 # DW_AT_GNU_all_call_sites
	.byte	25                      # DW_FORM_flag_present
	.byte	3                       # DW_AT_name
	.byte	14                      # DW_FORM_strp
	.byte	58                      # DW_AT_decl_file
	.byte	11                      # DW_FORM_data1
	.byte	59                      # DW_AT_decl_line
	.byte	11                      # DW_FORM_data1
	.byte	39                      # DW_AT_prototyped
	.byte	25                      # DW_FORM_flag_present
	.byte	63                      # DW_AT_external
	.byte	25                      # DW_FORM_flag_present
	.byte	0                       # EOM(1)
	.byte	0                       # EOM(2)
	.byte	3                       # Abbreviation Code
	.byte	5                       # DW_TAG_formal_parameter
	.byte	0                       # DW_CHILDREN_no
	.byte	2                       # DW_AT_location
	.byte	23                      # DW_FORM_sec_offset
	.byte	3                       # DW_AT_name
	.byte	14                      # DW_FORM_strp
	.byte	58                      # DW_AT_decl_file
	.byte	11                      # DW_FORM_data1
	.byte	59                      # DW_AT_decl_line
	.byte	11                      # DW_FORM_data1
	.byte	73                      # DW_AT_type
	.byte	19                      # DW_FORM_ref4
	.byte	0                       # EOM(1)
	.byte	0                       # EOM(2)
	.byte	4                       # Abbreviation Code
	.byte	52                      # DW_TAG_variable
	.byte	0                       # DW_CHILDREN_no
	.byte	2                       # DW_AT_location
	.byte	23                      # DW_FORM_sec_offset
	.byte	3                       # DW_AT_name
	.byte	14                      # DW_FORM_strp
	.byte	58                      # DW_AT_decl_file
	.byte	11                      # DW_FORM_data1
	.byte	59                      # DW_AT_decl_line
	.byte	11                      # DW_FORM_data1
	.byte	73                      # DW_AT_type
	.byte	19                      # DW_FORM_ref4
	.byte	0                       # EOM(1)
	.byte	0                       # EOM(2)
	.byte	5                       # Abbreviation Code
	.byte	55                      # DW_TAG_restrict_type
	.byte	0                       # DW_CHILDREN_no
	.byte	73                      # DW_AT_type
	.byte	19                      # DW_FORM_ref4
	.byte	0                       # EOM(1)
```

```
	.byte	0                       # EOM(2)
	.byte	6                       # Abbreviation Code
	.byte	15                      # DW_TAG_pointer_type
	.byte	0                       # DW_CHILDREN_no
	.byte	73                      # DW_AT_type
	.byte	19                      # DW_FORM_ref4
	.byte	0                       # EOM(1)
	.byte	0                       # EOM(2)
	.byte	7                       # Abbreviation Code
	.byte	22                      # DW_TAG_typedef
	.byte	0                       # DW_CHILDREN_no
	.byte	73                      # DW_AT_type
	.byte	19                      # DW_FORM_ref4
	.byte	3                       # DW_AT_name
	.byte	14                      # DW_FORM_strp
	.byte	58                      # DW_AT_decl_file
	.byte	11                      # DW_FORM_data1
	.byte	59                      # DW_AT_decl_line
	.byte	11                      # DW_FORM_data1
	.byte	0                       # EOM(1)
	.byte	0                       # EOM(2)
	.byte	8                       # Abbreviation Code
	.byte	36                      # DW_TAG_base_type
	.byte	0                       # DW_CHILDREN_no
	.byte	3                       # DW_AT_name
	.byte	14                      # DW_FORM_strp
	.byte	62                      # DW_AT_encoding
	.byte	11                      # DW_FORM_data1
	.byte	11                      # DW_AT_byte_size
	.byte	11                      # DW_FORM_data1
	.byte	0                       # EOM(1)
	.byte	0                       # EOM(2)
	.byte	0                       # EOM(3)
	.section	.debug_info,"",@progbits
.Lcu_begin0:
	.long	.Ldebug_info_end0-.Ldebug_info_start0 # Length of Unit
.Ldebug_info_start0:
	.short	4                       # DWARF version number
	.long	.debug_abbrev           # Offset Into Abbrev. Section
	.byte	8                       # Address Size (in bytes)
	.byte	1                       # Abbrev [1] 0xb:0xa7 DW_TAG_compile_unit
	.long	.Linfo_string0          # DW_AT_producer
	.short	12                      # DW_AT_language
	.long	.Linfo_string1          # DW_AT_name
	.long	.Lline_table_start0     # DW_AT_stmt_list
	.long	.Linfo_string2          # DW_AT_comp_dir
	.quad	.Lfunc_begin0           # DW_AT_low_pc
	.long	.Lfunc_end0-.Lfunc_begin0 # DW_AT_high_pc
	.byte	2                       # Abbrev [2] 0x2a:0x43 DW_TAG_subprogram
	.quad	.Lfunc_begin0           # DW_AT_low_pc
	.long	.Lfunc_end0-.Lfunc_begin0 # DW_AT_high_pc
	.byte	1                       # DW_AT_frame_base
	.byte	87
                                        # DW_AT_GNU_all_call_sites
	.long	.Linfo_string3          # DW_AT_name
	.byte	1                       # DW_AT_decl_file
	.byte	9                       # DW_AT_decl_line
                                        # DW_AT_prototyped
```

```
                                        # DW_AT_external
    .byte   3                           # Abbrev [3] 0x3f:0xf
DW_TAG_formal_parameter
    .long   .Ldebug_loc0                # DW_AT_location
    .long   .Linfo_string4              # DW_AT_name
    .byte   1                           # DW_AT_decl_file
    .byte   9                           # DW_AT_decl_line
    .long   109                         # DW_AT_type
    .byte   3                           # Abbrev [3] 0x4e:0xf
DW_TAG_formal_parameter
    .long   .Ldebug_loc1                # DW_AT_location
    .long   .Linfo_string8              # DW_AT_name
    .byte   1                           # DW_AT_decl_file
    .byte   9                           # DW_AT_decl_line
    .long   109                         # DW_AT_type
    .byte   4                           # Abbrev [4] 0x5d:0xf DW_TAG_variable
    .long   .Ldebug_loc2                # DW_AT_location
    .long   .Linfo_string9              # DW_AT_name
    .byte   1                           # DW_AT_decl_file
    .byte   10                          # DW_AT_decl_line
    .long   148                         # DW_AT_type
    .byte   0                           # End Of Children Mark
    .byte   5                           # Abbrev [5] 0x6d:0x5 DW_TAG_restrict_type
    .long   114                         # DW_AT_type
    .byte   6                           # Abbrev [6] 0x72:0x5 DW_TAG_pointer_type
    .long   119                         # DW_AT_type
    .byte   7                           # Abbrev [7] 0x77:0xb DW_TAG_typedef
    .long   130                         # DW_AT_type
    .long   .Linfo_string7              # DW_AT_name
    .byte   3                           # DW_AT_decl_file
    .byte   24                          # DW_AT_decl_line
    .byte   7                           # Abbrev [7] 0x82:0xb DW_TAG_typedef
    .long   141                         # DW_AT_type
    .long   .Linfo_string6              # DW_AT_name
    .byte   2                           # DW_AT_decl_file
    .byte   38                          # DW_AT_decl_line
    .byte   8                           # Abbrev [8] 0x8d:0x7 DW_TAG_base_type
    .long   .Linfo_string5              # DW_AT_name
    .byte   8                           # DW_AT_encoding
    .byte   1                           # DW_AT_byte_size
    .byte   7                           # Abbrev [7] 0x94:0xb DW_TAG_typedef
    .long   159                         # DW_AT_type
    .long   .Linfo_string12             # DW_AT_name
    .byte   3                           # DW_AT_decl_file
    .byte   27                          # DW_AT_decl_line
    .byte   7                           # Abbrev [7] 0x9f:0xb DW_TAG_typedef
    .long   170                         # DW_AT_type
    .long   .Linfo_string11             # DW_AT_name
    .byte   2                           # DW_AT_decl_file
    .byte   45                          # DW_AT_decl_line
    .byte   8                           # Abbrev [8] 0xaa:0x7 DW_TAG_base_type
    .long   .Linfo_string10             # DW_AT_name
    .byte   7                           # DW_AT_encoding
    .byte   8                           # DW_AT_byte_size
    .byte   0                           # End Of Children Mark
.Ldebug_info_end0:
    .ident  "clang version 10.0.0-4ubuntu1 "
    .section    ".note.GNU-stack","",@progbits
```

```
    .addrsig
    .section    .debug_line,"",@progbits
.Lline_table_start0:
```

内存未对齐。可以将所有元素移动一位来重新对齐b数组，或者使用偏移量。向量化后程序运行会更快。

Write-up 5: Check the assembly and verify that it does in fact vectorize properly. Also what do you notice when you run the command

```
$ clang -O3 example4.c -o example4; ./example4
```

with and without the `-ffast-math` flag? Specifically, why do you a see a difference in the output.

答:

对于汇编代码，材料已列举使用addsd命令的非向量化代码。

在makefile中添加-ffast-math

```
CC := clang

TARGETS = example1 example2 example3 example4

OTHER_CFLAGS ?= -ffast-math
CFLAGS := -Wall -g -std=gnu99

ifeq ($(DEBUG),1)
  CFLAGS += -O0
else
  CFLAGS += -O3
endif

V ?= 1

ifeq ($(VECTORIZE),1)
  #CFLAGS += -ftree-vectorizer-verbose=$(V)
  CFLAGS += -Rpass=loop-vectorize -Rpass-missed=loop-vectorize
else
  CFLAGS += -fno-vectorize
endif

ifeq ($(ASSEMBLE),1)
  CFLAGS += -S
endif

ifeq ($(AVX2),1)
  CFLAGS += -mavx2
endif

CFLAGS += $(OTHER_CFLAGS)

LDFLAGS := -lrt

# You shouldn't need to touch this.  This keeps track of whether or
# not you've changed CFLAGS.
```

重新编译，可以看到确实应用了向量化：

```
zhangkeer@ubuntu:~/a4/recitation3$ make clean; make ASSEMBLE=1 VECTORIZE=1 example4.o
rm -rf
rm -f *.o *.s .cflags perf.data *.lst
clang -Wall -g -std=gnu99 -O3 -Rpass=loop-vectorize -Rpass-missed=loop-vectorize -S -ffast-math -c example4.c
example4.c:17:3: remark: vectorized loop (vectorization width: 2, interleaved count: 2) [-Rpass=loop-vectorize]
  for (i = 0; i < SIZE; i++) {
  ^
example4.c:25:3: remark: the cost-model indicates that vectorization is not beneficial [-Rpass-missed=loop-vectorize]
  for (int i = 0; i < SIZE; i++) {
  ^
example4.c:25:3: remark: the cost-model indicates that interleaving is not beneficial [-Rpass-missed=loop-vectorize]
example4.c:17:3: remark: vectorized loop (vectorization width: 2, interleaved count: 2) [-Rpass=loop-vectorize]
  for (i = 0; i < SIZE; i++) {
  ^
zhangkeer@ubuntu:~/a4/recitation3$
```

汇编代码中的命令也由addsd变成了addpd：

```
        .text
        .file   "example4.c"
        .globl  test                    # -- Begin function test
        .p2align        4, 0x90
        .type   test,@function
test:                                   # @test
.Lfunc_begin0:
        .file   1 "/home/zhangkeer/a4/recitation3" "example4.c"
        .loc    1 10 0                  # example4.c:10:0
        .cfi_startproc
# %bb.0:
        #DEBUG_VALUE: test:a <- $rdi
        #DEBUG_VALUE: test:a <- $rdi
        xorpd   %xmm0, %xmm0
        xorl    %eax, %eax
.Ltmp0:
        #DEBUG_VALUE: test:i <- 0
        #DEBUG_VALUE: test:y <- 0.000000e+00
        #DEBUG_VALUE: test:x <- undef
        xorpd   %xmm1, %xmm1
.Ltmp1:
        .p2align        4, 0x90
.LBB0_1:                                # =>This Inner Loop Header: Depth=1
        #DEBUG_VALUE: test:y <- 0.000000e+00
        #DEBUG_VALUE: test:i <- 0
        #DEBUG_VALUE: test:a <- $rdi
        .loc    1 18 7 prologue_end     # example4.c:18:7
        addpd   (%rdi,%rax,8), %xmm0
        addpd   16(%rdi,%rax,8), %xmm1
        addpd   32(%rdi,%rax,8), %xmm0
        addpd   48(%rdi,%rax,8), %xmm1
        addpd   64(%rdi,%rax,8), %xmm0
        addpd   80(%rdi,%rax,8), %xmm1
        addpd   96(%rdi,%rax,8), %xmm0
        addpd   112(%rdi,%rax,8), %xmm1
.Ltmp2:
        .loc    1 17 26                 # example4.c:17:26
```

不添加编译时标志 -ffast-math

```
clang -O3 example4.c -o example4; ./example4
```

输出结果为：

```
zhangkeer@ubuntu:~/a4/recitation3$ clang -O3 example4.c -o example4; ./example4
The decimal floating point sum result is: 11.667578
The raw floating point sum result is: 0x1.755cccec10aa5p+3
zhangkeer@ubuntu:~/a4/recitation3$
```

添加编译时标志 -ffast-math

```
clang -O3 -ffast-math  example4.c -o example4; ./example4
```

输出结果为：

```
zhangkeer@ubuntu:~/a4/recitation3$ clang -O3 -ffast-math  example4.c -o example4
; ./example4
The decimal floating point sum result is: 11.667578
The raw floating point sum result is: 0x1.755cccec10aa3p+3
zhangkeer@ubuntu:~/a4/recitation3$
```

输出结果不同，因为我们设置标志位允许了 clang 重新排序我们给它的操作。

Write-up 6: What speedup does the vectorized code achieve over the unvectorized code? What additional speedup does using `-mavx2` give? You may wish to run this experiment several times and take median elapsed times; you can report answers to the nearest 100% (e.g., $2\times$, $3\times$, etc). What can you infer about the bit width of the default vector registers on the `awsrun` machines? What about the bit width of the AVX2 vector registers? *Hint*: aside from speedup and the vectorization report, the most relevant information is that the data type for each array is `uint32_t`.

Without vectorization:

```
$ make
$ time ./loop
```

运行结果:

```
zhangkeer@ubuntu:~/a4/homework3$ make
clang -Wall -std=gnu99 -g -O3 -DNDEBUG -fno-vectorize  -c loop.c
clang -o loop loop.o -lrt
zhangkeer@ubuntu:~/a4/homework3$ time ./loop
Elapsed execution time: 0.048396 sec; N: 1024, I: 100000, __OP__: +, __TYPE__: u
int32_t

real	0m0.050s
user	0m0.049s
sys	0m0.000s
zhangkeer@ubuntu:~/a4/homework3$
```

修改I的值，多次运行，使结果更有代表性：

```
zhangkeer@ubuntu:~/a4/homework3$ time ./loop
Elapsed execution time: 4.558956 sec; N: 1024, I: 10000000, __OP__: +, __TYPE__: uint32_t

real	0m4.560s
user	0m4.421s
sys	0m0.028s
zhangkeer@ubuntu:~/a4/homework3$
```

With vectorization：

```
$ make clean
$ make VECTORIZE=1
$ time ./loop
```

运行结果:

```
zhangkeer@ubuntu:~/a4/homework3$ make clean
rm -f loop *.o *.s .cflags perf.data */perf.data
zhangkeer@ubuntu:~/a4/homework3$ make VECTORIZE=1
clang -Wall -std=gnu99 -g -O3 -DNDEBUG -Rpass=loop-vectorize -Rpass-missed=loop-vectorize -ffast-math  -c loop.c
loop.c:70:9: remark: vectorized loop (vectorization width: 4, interleaved count: 2) [-Rpass=loop-vectorize]
        for (j = 0; j < N; j++) {
        ^
clang -o loop loop.o -lrt
zhangkeer@ubuntu:~/a4/homework3$ time ./loop
Elapsed execution time: 0.021711 sec; N: 1024, I: 100000, __OP__: +, __TYPE__: uint32_t

real	0m0.024s
user	0m0.024s
sys	0m0.000s
zhangkeer@ubuntu:~/a4/homework3$
```

修改I的值，多次运行，使结果更有代表性:

```
zhangkeer@ubuntu:~/a4/homework3$ time ./loop
Elapsed execution time: 1.133115 sec; N: 1024, I: 10000000, __OP__: +, __TYPE__: uint32_t

real	0m1.135s
user	0m1.134s
sys	0m0.000s
zhangkeer@ubuntu:~/a4/homework3$
```

With AVX instructions:

```
$ make clean
$ make VECTORIZE=1 AVX2=1
$ time ./loop
```

运行结果:

```
zhangkeer@ubuntu:~/a4/homework3$ make clean
rm -f loop *.o *.s .cflags perf.data */perf.data
zhangkeer@ubuntu:~/a4/homework3$ make VECTORIZE=1 AVX2=1
clang -Wall -std=gnu99 -g -O3 -DNDEBUG -Rpass=loop-vectorize -Rpass-missed=loop-vectorize -ffast-math -mavx2  -c loop.c
loop.c:70:9: remark: vectorized loop (vectorization width: 8, interleaved count: 4) [-Rpass=loop-vectorize]
        for (j = 0; j < N; j++) {
        ^
clang -o loop loop.o -lrt
zhangkeer@ubuntu:~/a4/homework3$ time ./loop
Elapsed execution time: 0.006121 sec; N: 1024, I: 100000, __OP__: +, __TYPE__: uint32_t

real	0m0.008s
user	0m0.004s
sys	0m0.004s
zhangkeer@ubuntu:~/a4/homework3$
```

修改I的值，多次运行，使结果更有代表性：

```
zhangkeer@ubuntu:~/a4/homework3$ time ./loop
Elapsed execution time: 0.597142 sec; N: 1024, I: 10000000, __OP__: +, __TYPE__: uint32_t

real	0m0.599s
user	0m0.594s
sys	0m0.005s
zhangkeer@ubuntu:~/a4/homework3$
```

执行lscpu的结果是：

```
Terminal                    Oct 20 02:07
zhangkeer@ubuntu: ~/a4/homework3
zhangkeer@ubuntu:~/a4/homework3$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   45 bits physical, 48 bits virtual
CPU(s):                          2
On-line CPU(s) list:             0,1
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       2
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz
Stepping:                        11
CPU MHz:                         1799.997
BogoMIPS:                        3599.99
Hypervisor vendor:               VMware
Virtualization type:             full
L1d cache:                       64 KiB
L1i cache:                       64 KiB
L2 cache:                        512 KiB
L3 cache:                        12 MiB
NUMA node0 CPU(s):               0,1
Vulnerability Itlb multihit:     KVM: Mitigation: VMX unsupported
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Vulnerable: Clear CPU buffers attempted, no mic
                                 rocode; SMT Host state unknown
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled v
                                 ia prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user
                                  pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB condit
                                 ional, IBRS_FW, STIBP disabled, RSB filling
Vulnerability Srbds:             Unknown: Dependent on hypervisor status
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtr
                                 r pge mca cmov pat pse36 clflush mmx fxsr sse s
                                 se2 ss syscall nx pdpe1gb rdtscp lm constant_ts
                                 c arch_perfmon nopl xtopology tsc_reliable nons
                                 top_tsc cpuid pni pclmulqdq ssse3 fma cx16 pcid
                                  sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline
                                 _timer aes xsave avx f16c rdrand hypervisor lah
                                 f_lm abm 3dnowprefetch cpuid_fault invpcid_sing
                                 le ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi
                                 1 avx2 smep bmi2 invpcid rdseed adx smap clflus
                                 hopt xsaveopt xsavec xgetbv1 xsaves arat flush_
```

综合上面的结果，向量化代码将非向量化代码的运行速度提升了约四倍，使用AVX2向量寄存器又在向量化代码的基础上加速了两倍左右。

由于每个数组元素的数据类型是uint32_t，默认向量寄存器的位宽应该是4×32=128（性能提升四倍），AVX2 向量寄存器的位宽应该是2×128=256（性能又提升两倍）。