



# Multiprocessors

Feb-Jun 2021

6

## Vectorization

Alejandro Guajardo Moreno



## Auto-vectorization Report

`/Qvec-report:{1}{2}`

### **`/Qvec-report:1`**

Outputs an informational message for loops that are vectorized.

### **`/Qvec-report:2`**

Outputs an informational message for loops that are vectorized and for loops that are not vectorized, together with a reason code.



# Auto-vectorization

Using /Qvec-report:2...

## Output

Show output from: Build

```
1>----- Rebuild All started: Project: Test1, Configuration: Release Win32 -----
1>Test1.c
1>Generating code
1>Previous IPDB not found, fall back to full compilation.
1>
1>--- Analyzing function: main
1>C:\Users\Alex\source\repos\Test1\Test1.c(13) : info C5001: loop vectorized
1>C:\Users\Alex\source\repos\Test1\Test1.c(17) : info C5001: loop vectorized
1>C:\Users\Alex\source\repos\Test1\Test1.c(20) : info C5002: loop not vectorized due to reason '1200'
1>All 4 functions were compiled because no usable IPDB/IOB from previous compilation was found.
1>Finished generating code
1>Test1.vcxproj -> C:\Users\Alex\source\repos\Test1\Release\Test1.exe
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====
|
```



## Auto-vectorization report (GCC)

`-fopt-info-vec[=file.ext]`

`-fopt-info-vec-missed[=file.ext]`

### **-fopt-info-vec**

Outputs an informational message for loops that are vectorized.

### **-fopt-info-vec-missed**

Outputs an informational message for loops that are vectorized and for loops that are not vectorized, together with a short explanation.





## Auto-vectorization report (GCC)

Using -fopt-info-vec & -fopt-info-vec-missed...

```
mp@multiprocesadores:~/multiprocs$ gcc autovec.c -o autovec -O -ftree-vectorize -mavx2
-fopt-info-vec -fopt-info-vec-missed
autovec.c:22:1: note: not vectorized: loop contains function calls or data references
that cannot be analyzed
autovec.c:15:1: note: not vectorized: loop contains function calls or data references
that cannot be analyzed
autovec.c:11:1: note: misalign = 0 bytes of ref B[i_30]
autovec.c:11:1: note: misalign = 0 bytes of ref A[i_30]
autovec.c:11:1: note: loop vectorized
autovec.c:6:5: note: not vectorized: not enough data-refs in basic block.
autovec.c:12:9: note: not vectorized: no vectype for stmt: MEM[(int *)vectp_B.4_7] = v
ect_vec_iv_.3_8;
    scalar_type: vector(8) int
autovec.c:12:9: note: not vectorized: no vectype for stmt: MEM[(int *)vectp_A.6_29] =
vect_vec_iv_.3_8;
    scalar_type: vector(8) int
autovec.c:12:9: note: not vectorized: no grouped stores in basic block.
autovec.c:19:1: note: not vectorized: no grouped stores in basic block.
autovec.c:15:1: note: not vectorized: not enough data-refs in basic block.
/usr/include/x86_64-linux-gnu/bits/stdio2.h:104:10: note: not vectorized: no grouped s
tores in basic block.
autovec.c:22:1: note: not vectorized: not enough data-refs in basic block.
autovec.c:6:5: note: not vectorized: not enough data-refs in basic block.
```



## Auto-vectorization – more GCC flags

### **-falign-functions=32**

Aligns the address of functions to be a multiple of 32 bytes.

### **-falign-loops=32**

Aligns the address of loops to be a multiple of 32 bytes.



## Auto-vectorization – GCC general optimizations

The `-O level` option to `gcc` turns on compiler optimization, when the specified value of `level` has the following effects:

0

The default reduces compilation time and has the effect that debugging always yields the expected result. This level is equivalent to not specifying the `-O` option at all. However, a number of optimization options are still enabled, for example: `-falign-loops`, `-finline-functions-called-once`, and `-fmove-loop-invariants`.

1

The compiler attempts to reduce both the size of the output binary code and the execution speed, but it does not perform optimizations that might substantially increase the compilation time.

2

The compiler performs optimizations that do not require a tradeoff of space for speed. Compared to level 1, level 2 optimization improves the performance of the output binary but it also increases the compilation time.

3

The compiler turns on the `-fgcse-after-reload`, `-finline-functions`, `-fipa-cp-clone`, `-fpredictive-commoning`, `-free-vectorize`, and `-funswitch-loops` options, which require a tradeoff of space for speed, in addition to the level 2 optimizations.

s

The compiler optimizes to reduce the size of the binary instead of execution speed.

Won't use them on this course, since we want to be able to know why a program is getting better.



## Auto-vectorization

### An inconvenient truth...

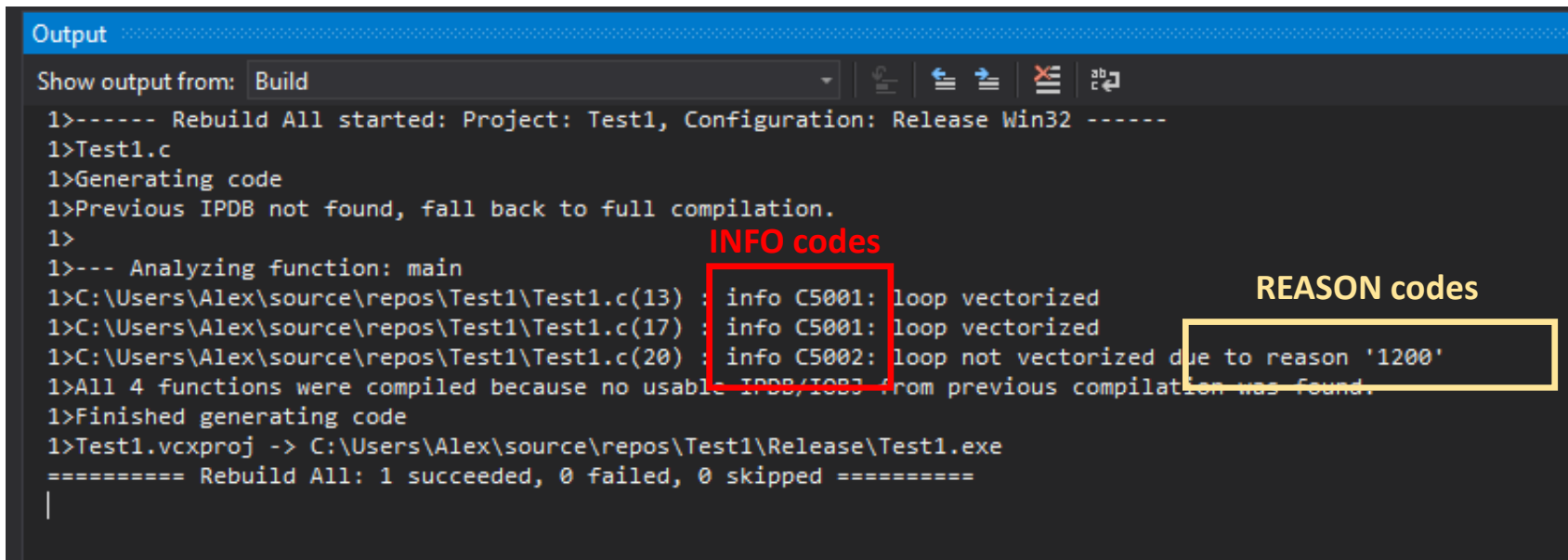
- Compilers are only partially successful at vectorizing.
- They play safe, using the info available to them.
- Programmer is the only one with the complete knowledge.
- When the compiler fails, programmers can
  - add compiler directives
  - apply loop transformations
- If after transforming the code, the compiler still fails to vectorize (or the performance of the generated code is poor), the only option is to program the vector extensions directly using intrinsics or assembly language.





## Auto-vectorization

INFO & REASON codes.



```
Output
Show output from: Build
1>----- Rebuild All started: Project: Test1, Configuration: Release Win32 -----
1>Test1.c
1>Generating code
1>Previous IPDB not found, fall back to full compilation.
1>
1>--- Analyzing function: main
1>C:\Users\Alex\source\repos\Test1\Test1.c(13) : info C5001: loop vectorized
1>C:\Users\Alex\source\repos\Test1\Test1.c(17) : info C5001: loop vectorized
1>C:\Users\Alex\source\repos\Test1\Test1.c(20) : info C5002: loop not vectorized due to reason '1200'
1>All 4 functions were compiled because no usable IPDB/IDB from previous compilation was found.
1>Finished generating code
1>Test1.vcxproj -> C:\Users\Alex\source\repos\Test1\Release\Test1.exe
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====
|
```

INFO codes

REASON codes



## Auto-vectorization

INFO codes.

5xxx



# Auto-vectorization

## INFO codes...

### Informational messages

Depending on the reporting level that you specify, one of the following informational messages appears for each loop.

For information about reason codes, refer to the next part of this article.

Informational Message	Description
5001	Loop vectorized.
5002	Loop not vectorized due to reason <code>**description**</code> .



## Auto-vectorization

**INFO** codes.

5xxx

**REASON** codes.

5xx

11xx

12xx

13xx

14xx – Incompatible flags or directives.

15xx – Memory aliasing.





# Auto-vectorization

## REASON codes 5xx...

Reason Code	Explanation
500	A generic message that covers several cases—for example, the loop includes multiple exits, or the loop header doesn't end by incrementing the induction variable.
501	<code>Induction variable is not local; or upper bound is not loop-invariant.</code>
502	<code>Induction variable is stepped in some manner other than a simple +1.</code>
503	<code>Loop includes exception-handling or switch statements.</code>
504	<code>Loop body may throw an exception that requires destruction of a C++ object.</code>

# Auto-vectorization

**REASON** code 500 - A generic message that covers several cases

```
int i = 0;
while (i < 1000)
{
    if (i == 4)
    {
        break;
    }
    ++i;
    A[i] = A[i] + 1;
}
```

- loop has non-vectorizable flow.
- "if", "break", "continue", the conditional operator "?", or function calls.
- The increment "++i" or "i++" must be the last statement in the loop.

```
for (int i = 0; i < 4; ++i)
{
    A[i] = A[i] + 1;
}
```

- ✓ Use a 'for' loop with single increment of induction variable.

Solution:



## Auto-vectorization

**REASON** code 501 Induction variable is not local; or upper bound is not loop-invariant.

```
int functionCalled() {  
    return rand();  
}  
  
int A[1000];  
for (int i = 0; i < functionCalled(); ++i)  
{  
    A[i] = A[i] + 1;  
}
```

- The compiler cannot discern the induction variable of this loop.
- The compiler cannot prove that the function call returns the same value each time.
- The compiler cannot prove that the call to does not modify the values of array A.

**Solution:**

```
for (int i = 0, imax = functionCalled(); i < imax; ++i)  
{  
    A[i] = A[i] + 1;  
}
```

- ✓ Make the variable local to the loop.



## Auto-vectorization

**REASON** code 502 Induction variable is stepped in some manner other than a simple +1.

```
for (int i = 0; i < 1000; ++i) {  
    A[i] = A[i] + 1;  
    ++i;  
  
    if (i < 100) {  
        ++i;  
    }  
}
```

- There are three increments to “i”
- Also, one increment is conditional.

**Solution:**

```
for (int i = 0; i < 1000; ++i)  
{  
    A[i] = A[i] + 1;  
}
```

- ✓ Ensure that there is just one increment of the induction variable,
- ✓ Placed the induction variable in the usual spot in the "for" loop.





## Auto-vectorization

**REASON** code 503 Loop includes exception-handling or switch statements.

```
for (int i = 0; i < 1000; ++i) {  
    switch (x) {  
        case 1: A[i] = A[i] + 1;  
        case 2: A[i] = A[i] + 2;  
        case 3: A[i] = A[i] + 3;  
        break;  
    }  
}
```

➤ There are inadmissible operations in the loop:

- exception handling
- switch statements

# Auto-vectorization

**REASON** code 503 Loop includes exception-handling or switch statements.

```
switch (x) {
  case 1:
    for (int i = 0; i < 1000; ++i)
      A[i] = A[i] + 1;
    break;
  case 2:
    for (int i = 0; i < 1000; ++i)
      A[i] = A[i] + 2;
    break;
  case 3:
    for (int i = 0; i < 1000; ++i)
      A[i] = A[i] + 3;
    break;
}
```

- ✓ Remove switch statements and exception handling constructs.
- ✓ In this case, instead of a SWITCH statement INSIDE a FOR loop, create three FOR loops INSIDE a SWITCH statement.

Solution:



## Auto-vectorization

**REASON** code 504 Loop body may throw an exception that requires destruction of a C++ object.

➤ Exception Handling flags are present in the compiler

Solution:

✓ Remove /Ehs and/or /Ehsc compiler flags.



# Auto-vectorization

**REASON** codes 11xx...

## 11xx reason codes

The 11xx reason codes apply to the vectorizer.

Reason Code	Explanation
1100	Loop contains control flow—for example, "if" or "?".
1101	Loop contains datatype conversion—perhaps implicit—that cannot be vectorized.
1102	Loop contains non-arithmetic or other non-vectorizable operations.
1103	Loop body includes shift operations whose size might vary within the loop.
1104	Loop body includes scalar variables.
1105	Loop includes a unrecognized reduction operation.
1106	Outer loop not vectorized.





## Auto-vectorization

**REASON** code 1100 Loop contains control flow—for example, "if" or "?".

```
for (int i = 0; i < 1000; ++i) {  
    if (x) {  
        A[i] = A[i] + 1;  
    }  
}
```

Flow control detected in the loop:

- "if"
- "?"
- Others like.

**Solution:**

```
for (int i = 0; i < 1000; ++i)  
{  
    A[i] = A[i] + 1;  
}
```

Flattening or removing control flow  
in the loop body.



## Auto-vectorization

**REASON** code 1101 Loop contains datatype conversion—perhaps implicit—that cannot be vectorized.

```
unsigned short A[1000], B[1000], C[1000], D[1000], in[1000];
```

```
    for (int j = 0; j < 1000; j++) {  
        float Gs = A[j] - B[j];  
        float Gc = C[j] - D[j];  
        in[j] = atan2f(Gs, Gc);  
    }
```

➤ Implicit conversion from float to int.

Solution:

- ✓ Use the correct data type.
- ✓ Check if you are enabling that types in the extension used.

\* Some compilers can find reason 1200 first.



## Auto-vectorization

**REASON** code 1102 Loop contains non-arithmetic or other non-vectorizable operations.

```
for (int i = 0; i < 1000; ++i)
{
    _rand32_step(&A[i]);
}
```

- The compiler is unable to vectorize an operation in the loop body:
  - Intrinsics
  - Non-arithmetic
  - Non-logical
  - Non-memory operation
- are not vectorizable.

Solution:

- ✓ Rewrite loop.



## Auto-vectorization

**REASON** code 1103 Loop body includes shift operations whose size might vary within the loop.

```
int A[1000], B[1000];

for (int i = 0; i < 1000; ++i)
{
    A[i] = A[i] >> B[i];
    int x = B[i];
    A[i] = A[i] >> x;
}
```

➤ Unable to vectorize a loop variant "shift" operation.

**Solution:**

```
int x = B[0];
for (int i = 0; i < 1000; ++i) {
    A[i] = A[i] >> x;
}
```

✓ Vectorizable only if shift amounts are loop invariant.





## Auto-vectorization

**REASON** code 1104 Loop body includes scalar variables.

```
int A[1000], B[1000];

int x;
for (int i = 0; i < 1000; ++i) {
    x = B[i];
    A[i] = A[i] + x;
}
return x;
```

- When it vectorizes a loop, the compiler must 'expand' scalar variables to a vector size such that they can fit in vector registers.
- In this example, x is tried to expand to be used in the vectorized loop. However, there is a use of 'x' beyond the loop body, which prohibits this expansion.

```
int x;
for (int i = 0; i < 1000; ++i) {
    x = B[i];
    A[i] = A[i] + x;
}
return 1;
```

- ✓ Limit scalars to be used only in the loop body and not beyond
- ✓ Keep their types consistent with the loop types.

Solution:



## Auto-vectorization

**REASON** code 1105 Loop includes a unrecognized reduction operation.

```
int A[1000];  
  
for (int i = 0, s = 0; i < 1000; ++i)  
{  
    s += A[i] + s;  
}
```

➤ The compiler cannot deduce the reduction pattern.

```
for (int i = 0; i < 1000; ++i)  
{  
    s += A[i];  
}
```

✓ Rewrite reduction condition.

\* Some compilers can find reason 1305 first.

Solution:



## Auto-vectorization

**REASON** code 1106 Outer loop not vectorized.

```
for (int i = 0; i < 1000; ++i)
{
    for (int j = 0; j < 1000; ++j)
    {
        A[j] = A[j] + 1;
    }
}
```

- Code 1106 is emitted when the compiler tries to vectorize an outer loop.
- Inner loops will vectorize.
- Outer loops won't vectorize.

```
for (int j = 0; j < 1000000; ++j) {
    A[j] = A[j] + 1;
}
```

- ✓ Try to separate the loops or fuse them.

Solution:



# Auto-vectorization

**REASON** codes 12xx...

## 12xx reason codes

The 12xx reason codes apply to the vectorizer.

Reason Code	Explanation
1200	Loop contains loop-carried data dependences that prevent vectorization. Different iterations of the loop interfere with each other such that vectorizing the loop would produce wrong answers, and the auto-vectorizer cannot prove to itself that there are no such data dependences.
1201	Array base changes during the loop.
1202	Field in a struct is not 32 or 64 bits wide.
1203	Loop body includes non-contiguous accesses into an array.



## Auto-vectorization

REASON code 1200

```
for (int i = 0; i < 1000; ++i)
{
    A[i] = A[i - 1] + 1;
}
```

➤ Data dependence.

✓ Rewrite the loop.

Solution:



## Auto-vectorization

REASON code 1201

```
for (int i = 0; i < 1000; ++i)
{
    A[i] = A[i] + 1;
    A++;
}
```

➤ An array base changes in the loop body

Solution:

✓ Rewriting your code so that varying the array base is not necessary.





## Auto-vectorization

REASON code 1202

```
struct S_1202 {  
    short a;  
    short b;  
} s[1000];  
  
void code_1202(S_1202* s) {  
    for (int i = 0; i < 1000; ++i) {  
        s[i].a = s[i].b + 1;  
    }  
}
```

- Code 1202 is emitted when non-vectorizable struct accesses are present in the loop body.
- 16 bit struct access is not vectorizable.

Solution:

- ✓ Only struct accesses that are 32 or 64 bits are vectorized.

# Auto-vectorization

REASON code 1203

```
for (int i = 0; i < 1000; ++i)
{
    A[i] += A[0] + 1;
    A[i] += A[i * 2 + 2] + 2;
}
```

➤ Constant memory access not vectorized.

➤ Non-contiguous memory access not vectorized

```
int temp = A[0];
for (int i = 0; i < 1000; ++i)
{
    A[i] += temp + 1;
}
```

✓ Don't use constant memory access.



Vectorization of some non-contiguous memory access is supported - for example, the gather/scatter pattern.

Solution:



# Auto-vectorization

**REASON** codes 13xx...

## 13xx reason codes

The 13xx reason codes apply to the vectorizer.

Reason Code	Explanation
1300	Loop body contains no-or very little-computation.
1301	Loop stride is not +1.
1302	Loop is a "do-while".
1303	Too few loop iterations for vectorization to provide value.
1304	Loop includes assignments that are of different sizes.
1305	Not enough type information.



## Auto-vectorization

**REASON** code 1300 Loop body contains no—or very little—computation.

```
for (int i = 0; i < 1000; ++i)
{
    A[i] = B[i];
}
```

➤ No computation in the loop body.

`memcpy(A, B, strlen(B) + 1);`

✓ Resolve using different approach.

Solution:



## Auto-vectorization

**REASON** code 1301 Loop stride is not +1.

```
for (int i = 0; i < 1000; i += 2)
{
    A[i] = A[i] + 1;
}
```

➤ Only loops that have a stride of positive 1 are vectorized.

```
for (int i = 0; i < 1000; i++)
{
    A[i] = A[i] + 1;
}
```

✓ Rewrite the loop.

Solution:

# Auto-vectorization

**REASON** code 1302 Loop is a "do-while".

```
int i = 0;
do
{
    A[i] = A[i] + 1;
} while (++i < 1000);
```

➤ Do-while loops are NOT vectorized.

```
int i = 0;
while (i < 1000) {
    A[i] = A[i] + 1;
    i++;
}
```

✓ Rewrite the loop. Eliminate do-whiles.



Only “while” and “for” loops are vectorized.

Solution:



# Auto-vectorization

**REASON** code 1303 Too few loop iterations for vectorization to provide value.

```
for (int i = 0; i < 5; ++i)
{
    A[i] = A[i] + 1;
}
```

- The number of iterations of the loop is too small to make vectorization profitable.

```
for (int i = 0; i < 4; ++i)
{
    A[i] = A[i] + 1;
}
```

- ✓ Not much that can be done.



EVEN IF the iterations are too small, if the loop computation fits perfectly in vector then the loop may be vectorized.

Solution:



## Auto-vectorization

**REASON** code 1304 Loop includes assignments that are of different sizes.

```
int* A, short* B
for (int i = 0; i < 1000; ++i)
{
    A[i] = A[i] + 1;
    B[i] = B[i] + 1;
}
```

➤ Different sized statements in the loop body.

```
int* A, short* B
for (int i = 0; i < 1000; ++i)
    A[i] = A[i] + 1;
for (int i = 0; i < 1000; ++i)
    B[i] = B[i] + 1;
```

✓ Separate the loops

\* Some compilers CAN vectorize this example.

Solution:



## Auto-vectorization

**REASON** code 1305 Not enough type information.

```
typedef struct S_1305 {  
    int a;  
    int b;  
} S_1305;  
void code_1305(S_1305* s, S_1305 x) {  
    for (int i = 0; i < 1000; ++i) {  
        s[i] = x;  
    }  
}
```

➤ The compiler can't discern proper vectorizable type information.

➤ Non-scalar loop types such as struct assignment.

Solution:

✓ Ensure that your loops have statements that operate on integers or floating point types.



# Auto-vectorization

**REASON** codes 14xx...

## 14xx reason codes

The 14xx reason codes occur when some option that is incompatible with vectorization is specified.

Reason Code	Explanation
1400	<code>#pragma loop(no_vector)</code> is specified.
1401	<code>/kernel switch</code> is specified when targeting x86 or ARM.
1402	<code>/arch:SSE2</code> or higher switch is not specified when targeting x86.
1403	<code>/arch:ATOM</code> switch is specified and the loop includes operations on doubles.
1404	<code>/O1</code> or <code>/Os</code> switch is specified.
1405	Vectorization is disabled to aid in dynamic-initializer-to-static-initializer optimization.



# Auto-vectorization

**REASON** codes 15xx...

## 15xx reason codes

The 15xx reason codes apply to aliasing. Aliasing occurs when a location in memory can be accessed by two different names.

Reason Code	Explanation
1500	Possible aliasing on multi-dimensional arrays.
1501	Possible aliasing on arrays-of-structs.
1502	Possible aliasing and array index is other than $n + K$ .
1503	Possible aliasing and array index has multiple offsets.
1504	Possible aliasing; would require too many runtime checks.
1505	Possible aliasing, but runtime checks are too complex.



## Auto-vectorization – Directives (Windows)

### #pragma loop(no vector)

```
// This loop will NOT be auto-vectorized
#pragma loop(no_vector)
for (int i = 0; i < SIZE; i++)
    A[i] = A[i] + B[i];
```



Will always generate a C5002 INFO with REASON 1400.

Use it to:

- Disable an Auto-vectorized loop with poor performance.
- Compare the performance of your code without and with vectorization.
- Be able to turn off auto-vectorization for some loops.



**Thank you!**

See you  
next time!

