5602 Assignment 2

Min Zhang – zhangm9@tcd.ie

November 2020

General directions

In this assignment we're going to explore vectorisation using two simple loop structures and our two favourite compilers, gcc and icc. We'll be making minor adjustments to the provided C file, vector_assignment.c, and passing different compiler flags. Don't forget to use the most recent version of gcc (version 9.2), not chuck's default version; you can load this by first running `module load cports`, then `module load gcc`.

For each numbered question below, indicate precisely (a) how you modified the code, and (b) how you invoked the compilers, either by specifying the single-line command you used or how you adjusted your makefile. (You don't need to describe how you returned anything in the code to its original form.) If you used a makefile, include this too.

Each of the questions require you to run the code, for both loop types, for one or both compilers. Produce a single table to organise all the timing results. You choose a loop type by passing "1" or "2" as the first command-line argument (have a look at the `switch` on line 59). You can measure performance for each loop type separately using the `time` command (let's take the "user" time, and round to three or four significant figures). For example, to time the second function, `loops_v2`, you would run

```
chaplinc@chuck vectors]$ time ./intelbinary 2
```

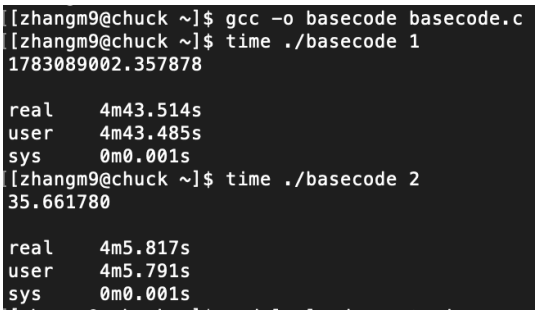
if you've named your binary `intelbinary`. The code returns a result to stderr, which is distinct for the two loop functions.

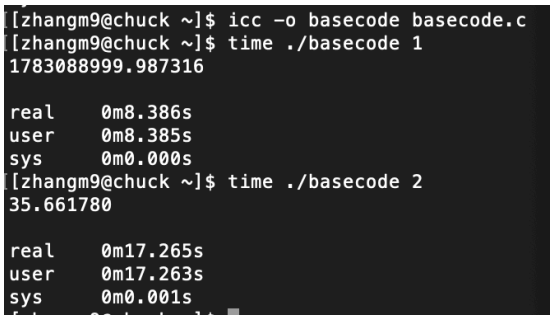
```
[zhangm9@chuck ~]$ module load cports gcc
[zhangm9@chuck ~]$ module load intel
```

Q1

Compile with default options (no optimisation or architecture flags) for both gcc and icc.

GCC code	gcc -o basecode basecode.c
Loop type	time ./basecode 1
Return value	1783089002.357878
User time	4m43.485s
Loop type	time ./basecode 2
Return value	35.661780
User time	4m5.791s



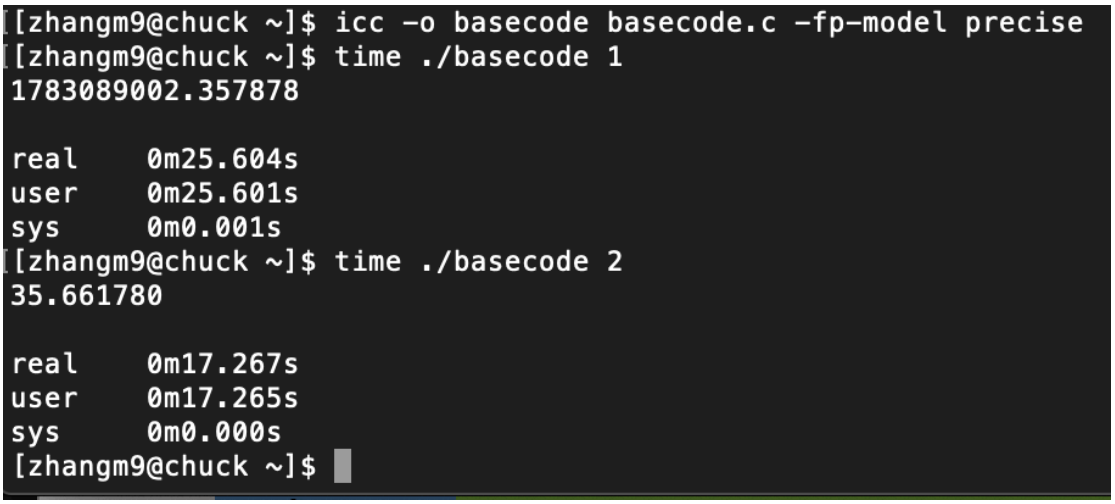


ICC code	icc -o basecode basecode.c
Loop type	time ./basecode 1
Return value	1783088999.987316
User time	0m8.385s
Loop type	time ./basecode 2
Return value	35.661780
User time	0m17.263s

Q2

Are the two compilers producing exactly the same result for the two loop functions? If not, what's the fractional discrepancy, and what do you think might be causing this? Try adding the flag `-fp-model precise` (yes, that is a space!) to your icc invocation. What effect does this have on speed and the returned result? If the returned results are now the same you should leave this flag in place when calling icc.

ICC code	icc -o basecode basecode.c -mode -fpl precise
Loop type	time ./basecode 1
Return value	1783089002.357878
User time	0m24.601s
Loop type	time ./basecode 2
Return value	35.661780
User time	0m17.265s



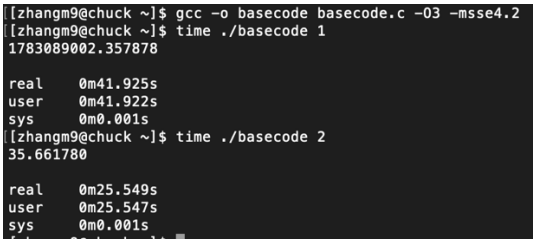
Answer:

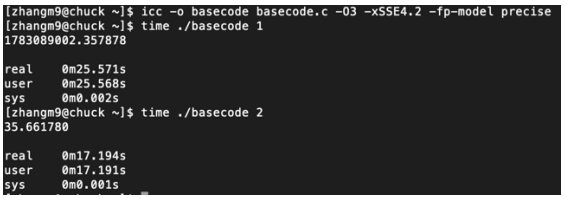
By default, the default mechanism for computer compilation is much more ICC than GCC compilation, and these additional operations make Float types somewhat polluting. As you can see from the results generated above, ICC is compiled differently from GCC, and Loop2 is compiled the same. After adding -FP-Model percise, we found that the compilation speed decreased significantly. It has no effect on the second cycle. While - FP-Model Precise: This has a certain optimization effect on Float security.

Q3

Compile with gcc and icc, using `-O3` and the correct architecture or vector-extension flag for chuck.

GCC code	gcc -o basecode basecode.c -O3 -msse4.2
Loop type	time ./basecode 1
Return value	1783089002.357878
User time	0m41.922s
Loop type	time ./basecode 2
Return value	35.661780
User time	0m25.547s



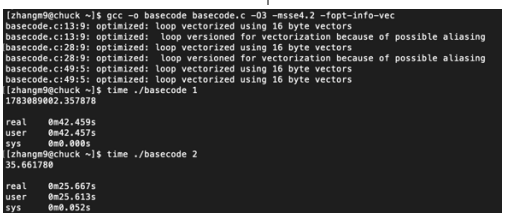


ICC code	icc -o basecode basecode.c -O3 -xSSE4.2 -fp-model precise
Loop type	time ./basecode 1
Return value	1783089002.357878
User time	0m25.568s
Loop type	time ./basecode 2
Return value	35.661780
User time	0m17.191s

Q4

Produce vectorisation reports with each compiler. Are the inner loops inside the two functions (lines 13 and 28 in the original C file) being vectorised? What about the loop in main (line 49)?

GCC code	gcc -o basecode basecode.c -O3 -fopt-info-vec -msse4.2
Loop type	time ./basecode 1
User time	0m42.457s
Return value	1783089002.357878
Loop type	time ./basecode 2
User time	0m25.613s
Return value	35.661780



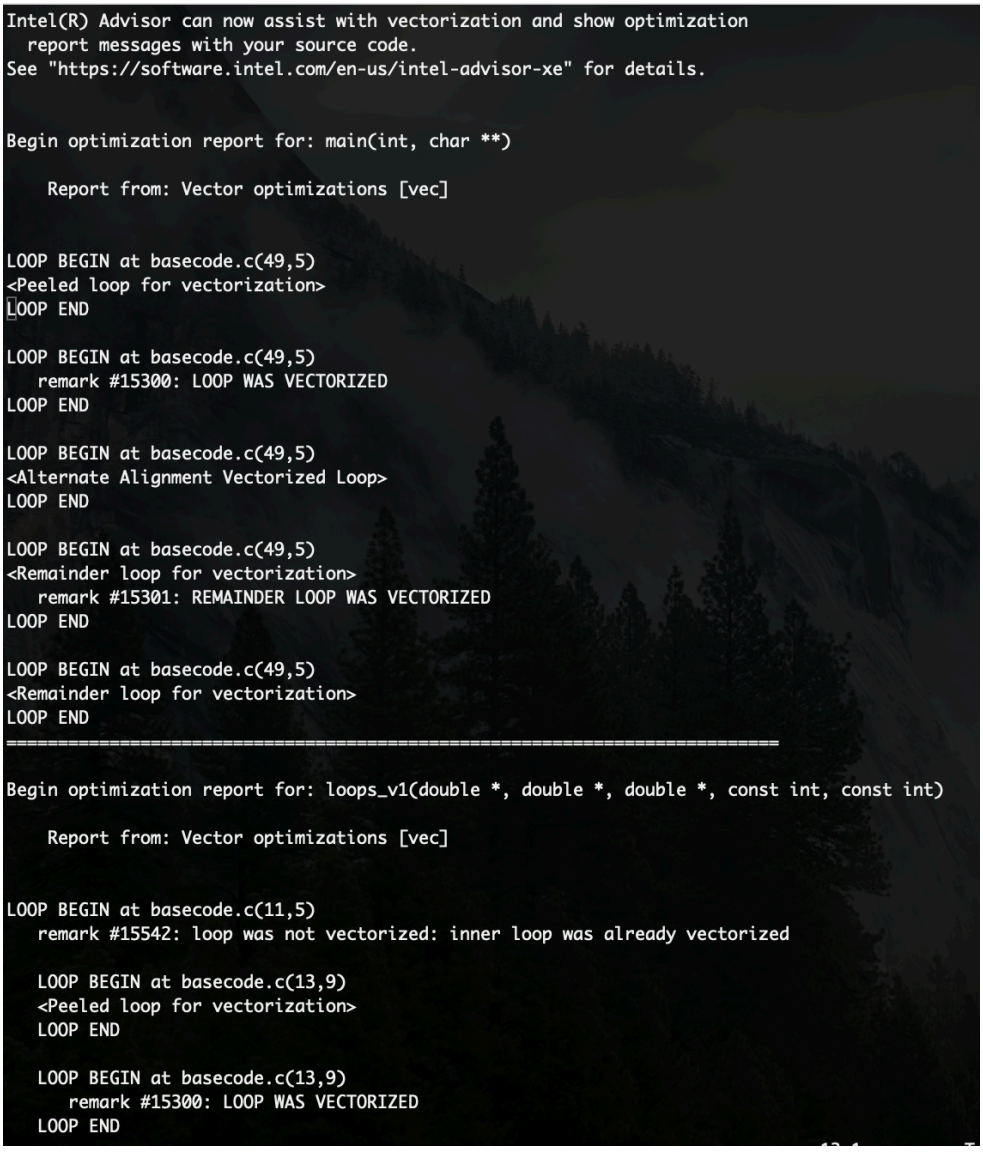
ICC code	icc -o basecode basecode.c -O3 -xSSE4.2 -qopt-report -qopt-report-phase=vec -fp-model precise vi basecode.optrpt
Loop type	time ./basecode 1
Return value	1783089002.357878
User time	0m25.595s
Loop type	time ./basecode 2
Return value	35.661780
User time	0m17.202s

GCC code	gcc -o basecode basecode.c -O3 -msse4.2 -fopt-info-vec
Line 13,line 28 and line 49 are being vectorized.	basecode.c:13:9: optimized: loop vectorized using 16 byte vectors basecode.c:13:9: optimized: loop versioned for vectorization because of possible aliasing basecode.c:28:9: optimized: loop vectorized using 16 byte vectors basecode.c:28:9: optimized: loop versioned for vectorization because of possible aliasing basecode.c:49:5: optimized: loop vectorized using 16 byte vectors basecode.c:49:5: optimized: loop vectorized using 16 byte vectors

```
[zhangm9@chuck ~]$ gcc -o basecode basecode.c -O3 -msse4.2 -fopt-info-vec
basecode.c:13:9: optimized: loop vectorized using 16 byte vectors
basecode.c:13:9: optimized:  loop versioned for vectorization because of possibl
e aliasing
basecode.c:28:9: optimized: loop vectorized using 16 byte vectors
basecode.c:28:9: optimized:  loop versioned for vectorization because of possibl
e aliasing
basecode.c:49:5: optimized: loop vectorized using 16 byte vectors
basecode.c:49:5: optimized: loop vectorized using 16 byte vectors
[zhangm9@chuck ~]$
```

ICC code	vi basecode.optrpt
Line 13,line 28 and line 49 are being vectorised.	LOOP BEGIN at basecode.c(49,5) <Peeled loop for vectorization> LOOP END LOOP BEGIN at basecode.c(49,5) remark #15300: LOOP WAS VECTORIZED LOOP END LOOP BEGIN at basecode.c(49,5) <Alternate Alignment Vectorized Loop> LOOP END LOOP BEGIN at basecode.c(49,5) <Remainder loop for vectorization> remark #15301: REMAINDER LOOP WAS VECTORIZED LOOP END LOOP BEGIN at basecode.c(49,5) <Remainder loop for vectorization> LOOP END ========== Begin optimization report for: loops_v1(double *, double *, double *, const int, const int)

	Report from: Vector optimizations [vec] LOOP BEGIN at basecode.c(11,5) remark #15542: loop was not vectorized: inner loop was already vectorized LOOP BEGIN at basecode.c(13,9) <Peeled loop for vectorization> LOOP END LOOP BEGIN at basecode.c(13,9) remark #15300: LOOP WAS VECTORIZED LOOP END



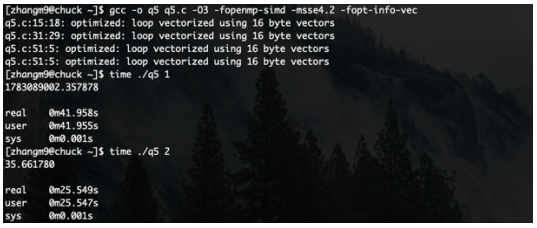
Q5

What changes when you add `#pragma omp simd` to the two tight inner loops (lines 13, 28)? Does this affect the vectorisation reports or the timing? You can leave these pragmas in place from now on. (Hint: don't forget any necessary compiler flags!)

```
/* This attribute asks the compiler not to inline the function  *
 * --- we don't want this adding extra optimisations here.      */
__attribute__ ((noinline))
double loops_v1(double *a, double *b, double *c, const int N, const int R)
{

    for (int r=0; r<R; r++)
    {
        #pragma omp simd
        for (int i=0; i<N; i++)
            a[i] += 0.027 * b[i] + c[i];
    }

    return a[N-1];
}
```

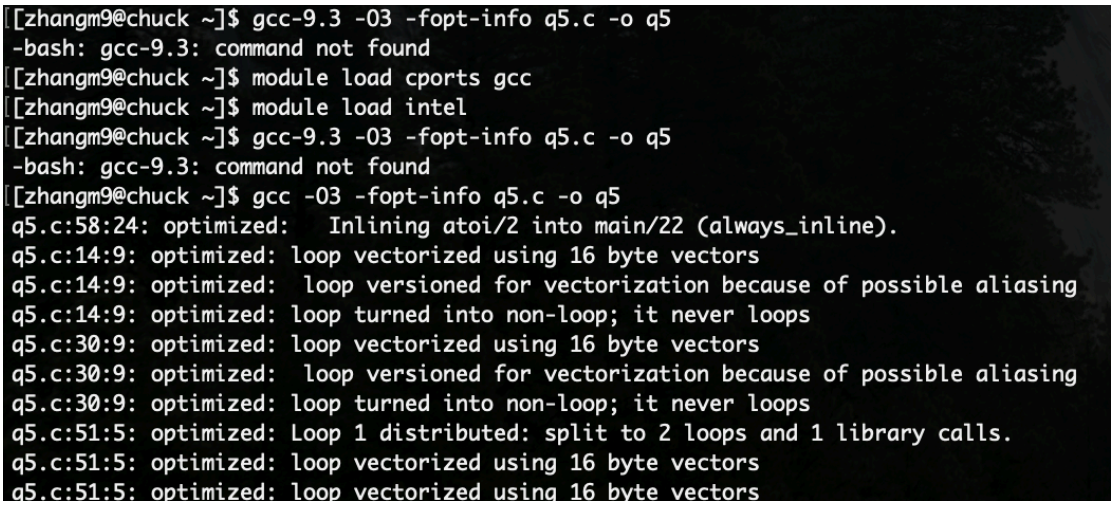
```
__attribute__ ((noinline))
double loops_v2(double *a, double *b, double *c,
                const int test_num, const int N, const int R)
{

    for (int r=0; r<R; r++)
    {
        #pragma omp simd
        for (int i=0; i<N; i++)
            a[i] = 0.027 * b[i] + c[i];

        /* To prevent the compiler from taking a McClane optimisation */
        if (a[N-1] < test_num-2)
            a[0] += 1.0;
    }

    return a[N-1];
}
```

GCC code	gcc -o q5 q5.c -O3 -fopenmp-simd -msse4.2 -fopt-info-vec
Loop type	time ./q5 1
Return value	0m41.955s
User time	1783089002.357878
Loop type	time ./q5 2
Return value	35.661780
User time	0m25.547s



ICC code	icc -o q5 q5.c -O3 -xSSE4.2 -qopt-report-phase=vec -qopenmp-simd -fp-model precise -qopt-report
Loop type	time ./q5 1
Return value	1783089002.357878
User time	0m25.567s
Loop type	time ./q5 2
Return value	35.661780
User time	0m17.164s

GCC code	gcc -O3 -fopt-info q5.c -o q5
Line 14 and line 30 are being vectorized.	q5.c:58:24: optimized: Inlining atoi/2 into main/22 (always_inline). q5.c:14:9: optimized: loop vectorized using 16 byte vectors q5.c:14:9: optimized: loop versioned for vectorization because of possible aliasing q5.c:14:9: optimized: loop turned into non-loop; it never loops q5.c:30:9: optimized: loop vectorized using 16 byte vectors q5.c:30:9: optimized: loop versioned for vectorization because of possible aliasing q5.c:30:9: optimized: loop turned into non-loop; it never loops q5.c:51:5: optimized: Loop 1 distributed: split to 2 loops and 1 library calls. q5.c:51:5: optimized: loop vectorized using 16 byte vectors q5.c:51:5: optimized: loop vectorized using 16 byte vectors



ICC code	vi q5.optrpt
Line 14 and line 19 are being vectorised.	Intel(R) Advisor can now assist with vectorization and show optimization report messages with your source code. See "https://software.intel.com/en-us/intel-advisor-xe" for details. Begin optimization report for: main(int, char **) Report from: Vector optimizations [vec] LOOP BEGIN at q5.c(51,5) <Peeled loop for vectorization> LOOP END

```
LOOP BEGIN at q5.c(51,5)
    remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Alternate Alignment Vectorized Loop>
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
    remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
LOOP END
```

```
Begin optimization report for: loops_v1(double *, double *, double *, const int, const int)

      Report from: Vector optimizations [vec]


LOOP BEGIN at q5.c(11,5)
    remark #15542: loop was not vectorized: inner loop was already vectorized

    LOOP BEGIN at q5.c(14,19)
    <Peeled loop for vectorization>
    LOOP END

    LOOP BEGIN at q5.c(14,19)
        remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
    LOOP END

    LOOP BEGIN at q5.c(14,19)
    <Alternate Alignment Vectorized Loop>
    LOOP END

    LOOP BEGIN at q5.c(14,19)
    <Remainder loop for vectorization>
        remark #15301: REMAINDER LOOP WAS VECTORIZED
    LOOP END

    LOOP BEGIN at q5.c(14,19)
```

```
    <Remainder loop for vectorization>
    LOOP END
LOOP END
```

```
OP BEGIN at q5.c(51,5)
<Peeled loop for vectorization>
LOOP END

LOOP BEGIN at q5.c(51,5)
    remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Alternate Alignment Vectorized Loop>
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
    remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
LOOP END
```

```
Begin optimization report for: loops_v1(double *, double *, double *, const int, const int)

      Report from: Vector optimizations [vec]


LOOP BEGIN at q5.c(11,5)
    remark #15542: loop was not vectorized: inner loop was already vectorized

    LOOP BEGIN at q5.c(14,19)
    <Peeled loop for vectorization>
    LOOP END

    LOOP BEGIN at q5.c(14,19)
        remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
    LOOP END
Begin optimization report for: loops_v2(double *, double *, double *, const int, const int, const int)

      Report from: Vector optimizations [vec]
```

```
LOOP BEGIN at q5.c(27,5)
    remark #15542: loop was not vectorized: inner loop was already vectorized

    LOOP BEGIN at q5.c(30,19)
    <Peeled loop for vectorization>
    LOOP END

    LOOP BEGIN at q5.c(30,19)
        remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
    LOOP END

    LOOP BEGIN at q5.c(30,19)
    <Alternate Alignment Vectorized Loop>
    LOOP END

    LOOP BEGIN at q5.c(30,19)
    <Remainder loop for vectorization>
        remark #15301: REMAINDER LOOP WAS VECTORIZED
    LOOP END

    LOOP BEGIN at q5.c(30,19)
    <Remainder loop for vectorization>
    LOOP END
LOOP END
```

```
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.


Begin optimization report for: main(int, char **)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q5.c(51,5)
<Peeled loop for vectorization>
LOOP END

LOOP BEGIN at q5.c(51,5)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Alternate Alignment Vectorized Loop>
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
LOOP END
===========================================================================
```

```
===========================================================================

Begin optimization report for: loops_v1(double *, double *, double *, const int, const int)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q5.c(11,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at q5.c(14,19)
   <Peeled loop for vectorization>
   LOOP END

   LOOP BEGIN at q5.c(14,19)
      remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at q5.c(14,19)
   <Alternate Alignment Vectorized Loop>
   LOOP END

   LOOP BEGIN at q5.c(14,19)
   <Remainder loop for vectorization>
      remark #15301: REMAINDER LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at q5.c(14,19)
   <Remainder loop for vectorization>
   LOOP END
LOOP END
===========================================================================
```

```
================================================================================

OP BEGIN at q5.c(51,5)
<Peeled loop for vectorization>
LOOP END

LOOP BEGIN at q5.c(51,5)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Alternate Alignment Vectorized Loop>
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q5.c(51,5)
<Remainder loop for vectorization>
LOOP END
================================================================================
```

```
====================================================================

Begin optimization report for: loops_v1(double *, double *, double *, const int, const int)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q5.c(11,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at q5.c(14,19)
   <Peeled loop for vectorization>
   LOOP END

   LOOP BEGIN at q5.c(14,19)
      remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   LOOP END
Begin optimization report for: loops_v2(double *, double *, double *, const int, const int, const int)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q5.c(27,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at q5.c(30,19)
   <Peeled loop for vectorization>
   LOOP END

   LOOP BEGIN at q5.c(30,19)
      remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at q5.c(30,19)
   <Alternate Alignment Vectorized Loop>
   LOOP END

   LOOP BEGIN at q5.c(30,19)
   <Remainder loop for vectorization>
      remark #15301: REMAINDER LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at q5.c(30,19)
   <Remainder loop for vectorization>
   LOOP END
LOOP END
====================================================================
```

Q6

With the pragmas in place, try compiling with icc without the `-fp-model precise` flag. What effect does this have on performance and the returned floating-point result? If the icc binary now gives the same returned result as gcc, you can leave out the `fp-model` flag from now on.

ICC code	icc -o q5 q5.c -qopenmp-simd -O3 -xSSE4.2
Loop type	time ./q5 1
Return value	1783089002.357878
User time	0m25.568s
Loop type	time ./q5 2
Return value	35.661780
User time	0m17.190s

```
[zhangm9@chuck ~]$ icc -o q5 q5.c -qopenmp-simd -O3 -xSSE4.2
[zhangm9@chuck ~]$ time ./q5 1
1783089002.357878

real    0m25.572s
user    0m25.568s
sys     0m0.002s
[zhangm9@chuck ~]$ time ./q5 2
35.661780

real    0m17.193s
user    0m17.190s
sys     0m0.002s
```

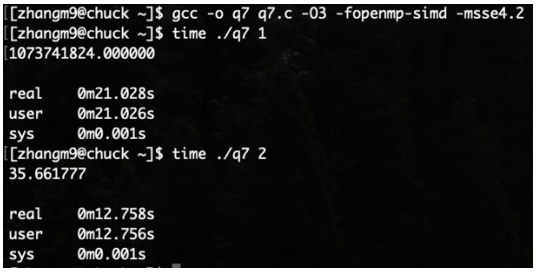
Answer:

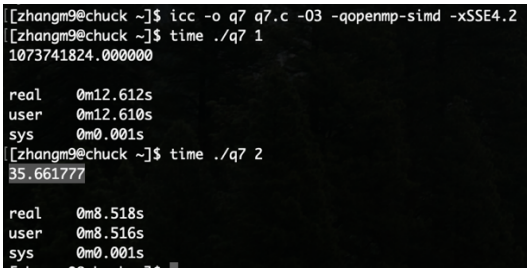
It can be seen from the results that after removing the -FP-Model, the optimization mechanism is in a silent state, and this result causes the compiler to ensure the security of Float value, so that it will not be affected. In terms of time expenditure, the two are the same.

Q7

So far we've been using 64-bit (double precision) floating point. Replace this with 32-bit floating point everywhere. Is there a significant speed-up? Is this what you would have expected, and why? (Reminder: in C, explicit floating-point numbers written directly into the code (known as literals) are doubles by default. A *single-precision* number is written as, e.g., `0.027f`, with a trailing "f".)

GCC code	gcc -o q7 q7.c -O3 -fopenmp-simd -msse4.2
Loop type	time ./q7 1
Return value	1073741824.000000
User time	0m21.026s
Loop type	time ./q7 2
Return value	35.661777
User time	0m12.756s





ICC code	icc -o q7 q7.c -O3 -qopenmp-simd -xSSE4.2
Loop type	time ./q7 1
Return value	1073741824.000000
User time	0m12.610s
Loop type	time ./q7 2
Return value	35.661777
User time	0m8.516s

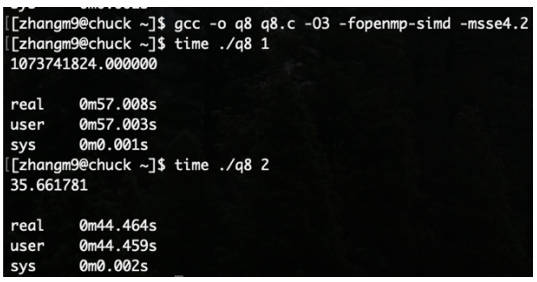
Answer:

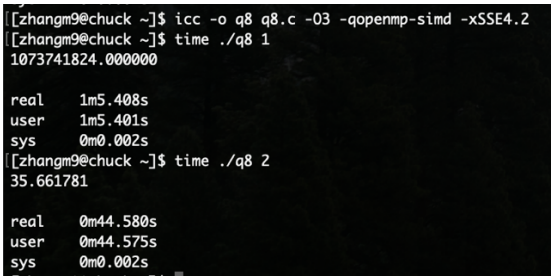
From the results of the two, there is a difference in speed. In ICC, the compilation speed has been significantly improved, as can be seen from the accuracy, but floating-point data is carried out much faster than dual-floating-point data.

Q8

What would the timing have been if you had forgotten to switch the floating-point literals to 32-bit numbers?

GCC code	gcc -o q8 q8.c -O3 -fopenmp-simd -msse4.2
Loop type	time ./q8 1
Return value	1073741824.000000
User time	0m57.003s
Loop type	time ./q8 2
Return value	35.661781
User time	0m44.459s





ICC code	icc -o q8 q8.c -O3 -qopenmp-simd -xSSE4.2
Loop type	time ./q8 1
Return value	1073741824.000000
User time	1m5.401s
Loop type	time ./q8 2
Return value	35.661781
User time	0m44.575s

Its output uses a long floating point type

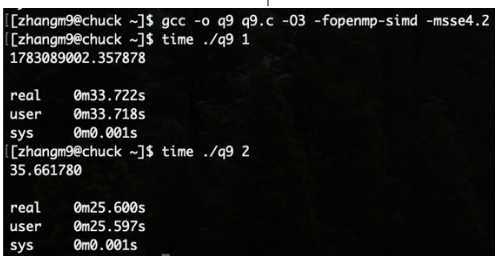
Q9

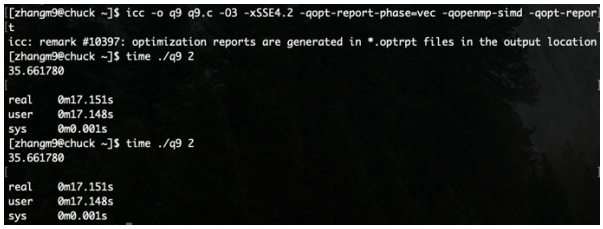
Switch back to 64-bit floating point everywhere. Now align the dynamically allocated memory, to the byte boundary appropriate for chuck. Check the timing both with the standard pragmas you've been using, and then with an extended pragma which includes a clause asserting that the memory is aligned. How does icc's vectorisation report change when the additional clauses are added?

```
/* This attribute asks the compiler not to inline the function  *
 * --- we don't want this adding extra optimisations here.      */
__attribute__ ((noinline))
double loops_v1(double *a, double *b, double *c, const int N, const int R)
{

    for (int r=0; r<R; r++)
    {
        #pragma omp simd aligned(a,b,c:16)
        for (int i=0; i<N; i++)
            a[i] += 0.027 * b[i] + c[i];
    }

    return a[N-1];
}
```

```
__attribute__ ((noinline))
double loops_v2(double *a, double *b, double *c,
                const int test_num, const int N, const int R)
{

    for (int r=0; r<R; r++)
    {
        #pragma omp simd aligned(a,b,c:16)
        for (int i=0; i<N; i++)
            a[i] = 0.027 * b[i] + c[i];

        /* To prevent the compiler from taking a McClane optimisation */
        if (a[N-1] < test_num-2)
            a[0] += 1.0;
    }

    return a[N-1];
}
```

GCC code	gcc -o q9 q9.c -O3 -fopenmp-simd -msse4.2
Loop type	time ./q9 1
Return value	1783089002.357878
User time	0m33.718s
Loop type	time ./q9 2
Return value	35.661781
User time	0m25.597s



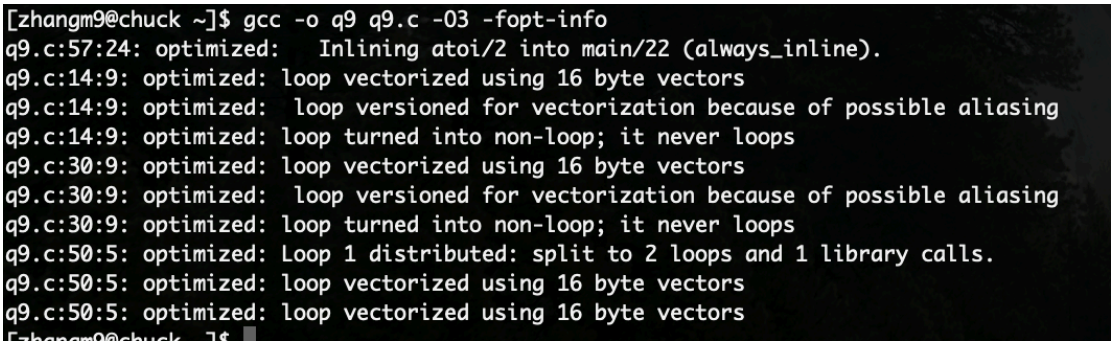


ICC code	icc -o q9 q9.c -O3 -xSSE4.2 -qopt-report-phase=vec -qopenmp-simd -qopt-report
Loop type	time ./q9 1
Return value	1783089002.357878
User time	0m25.453s
Loop type	time ./q9 2
Return value	35.661780
User time	0m17.149s

Answer:

While Icc is optimized for Intel processors, GCC is a general-purpose platform. However, it is not hard to see from the screenshot above that there is a significant difference in timing. This time difference is also consistent with the ICC and GCC differences we mentioned above. ICC is more efficient than GCC. But in the screenshots of the report, it is clear that the GCC does significantly more optimization items than the ICC does. In particular, most optimizations have been made for inline functions, since GCC was the first to support inline functions.

GCC code	gcc -o q9 q9.c -O3 -fopt-info
Line 14 and line 30	q9.c:57:24: optimized: Inlining atoi/2 into main/22 (always_inline). q9.c:14:9: optimized: loop vectorized using 16 byte vectors q9.c:14:9: optimized: loop versioned for vectorization because of possible aliasing q9.c:14:9: optimized: loop turned into non-loop; it never loops q9.c:30:9: optimized: loop vectorized using 16 byte vectors q9.c:30:9: optimized: loop versioned for vectorization because of possible aliasing q9.c:30:9: optimized: loop turned into non-loop; it never loops q9.c:50:5: optimized: Loop 1 distributed: split to 2 loops and 1 library calls. q9.c:50:5: optimized: loop vectorized using 16 byte vectors q9.c:50:5: optimized: loop vectorized using 16 byte vectors



ICC code	vi q9.optrpt
Line 14 and line 30	LOOP BEGIN at q9.c(50,5) <Peeled loop for vectorization> LOOP END LOOP BEGIN at q9.c(50,5) remark #15300: LOOP WAS VECTORIZED LOOP END LOOP BEGIN at q9.c(50,5) <Alternate Alignment Vectorized Loop> LOOP END LOOP BEGIN at q9.c(50,5) <Remainder loop for vectorization> remark #15301: REMAINDER LOOP WAS VECTORIZED LOOP END LOOP BEGIN at q9.c(50,5) <Remainder loop for vectorization>

```
LOOP END
```

```
Begin optimization report for: loops_v1(double *, double *, double *, const int, const int)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q9.c(11,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at q9.c(14,19)
      remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   LOOP END
LOOP END
```

```
Begin optimization report for: loops_v2(double *, double *, double *, const int, const int, const int)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q9.c(27,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at q9.c(30,19)
      remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   LOOP END
LOOP END
```

```
Intel(R) Advisor can now assist with vectorization and show optimization
  report messages with your source code.
See "https://software.intel.com/en-us/intel-advisor-xe" for details.


Begin optimization report for: main(int, char **)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q9.c(50,5)
<Peeled loop for vectorization>
LOOP END

LOOP BEGIN at q9.c(50,5)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q9.c(50,5)
<Alternate Alignment Vectorized Loop>
LOOP END

LOOP BEGIN at q9.c(50,5)
<Remainder loop for vectorization>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at q9.c(50,5)
<Remainder loop for vectorization>
LOOP END
===========================================================================

Begin optimization report for: loops_v1(double *, double *, double *, const int, const int)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q9.c(11,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at q9.c(14,19)
      remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   LOOP END
LOOP END
===========================================================================
```

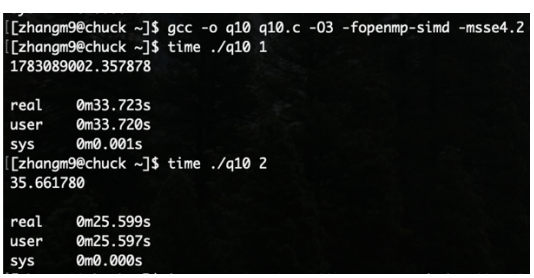
```
Begin optimization report for: loops_v2(double *, double *, double *, const int, const int, con
st int)

    Report from: Vector optimizations [vec]


LOOP BEGIN at q9.c(27,5)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at q9.c(30,19)
      remark #15301: OpenMP SIMD LOOP WAS VECTORIZED
   LOOP END
LOOP END
===========================================================================
```

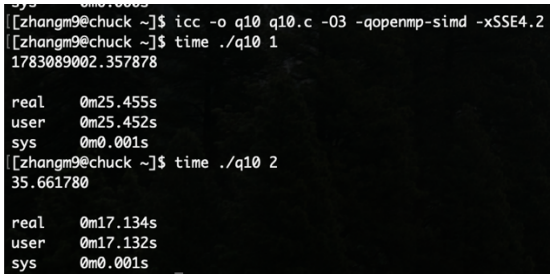
Q10

Replace the dynamically allocated memory with statically allocated (stack) arrays, also aligned to the relevant byte boundary. How does the performance compare?

```
const int N = 1024;      // Array length
const int R = (int)5e7; // Number of repetitions, to get better timing

double a[1024];
double b[1024];
double c[1024];
```

GCC code	gcc -o q10 q10.c -O3 -fopenmp-simd -msse4.2
Loop type	time ./q10 1
Return value	1783089002.357878
User time	0m33.720s
Loop type	time ./q10 2
Return value	35.661780
User time	0m25.597s





ICC code	icc -o q10 q10.c -O3 -qopenmp-simd -xSSE4.2
Loop type	time ./q10 1
Return value	1783089002.357878
User time	0m25.452s
Loop type	time ./q10 2
Return value	35.661780
User time	0m17.132s

Answer:

There is no significant difference in time performance

First, there is a significant difference between dynamic memory and static memory. The difference is mainly in whether the processor resources are occupied, the area where the resources are allocated, and the stages of allocation.

Static memory when the program starts running, first is to complete the assigned directly by the compiler, so when the program is run, is not dynamic CPU resources and dynamic memory on the contrary, it is in the process of program run dynamic allocation and recovery, with its the CPU utilization scheme at a disadvantage. Static memory is automatically managed by the system. Dynamic memory, on the other hand, requires the assistance of a programmer.

First, the memory has been byte paired, and it can cache the data as fast as possible on the CPU, depending on whether the compiler has the byte paired turned on. So this layer can exclude the impact of alignment.
Therefore, static memory has no impact on the performance of the program while dynamic memory always affects the performance of the program.

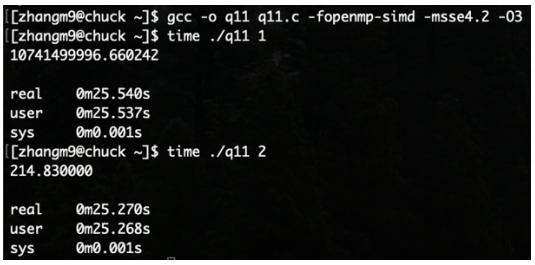
Q11

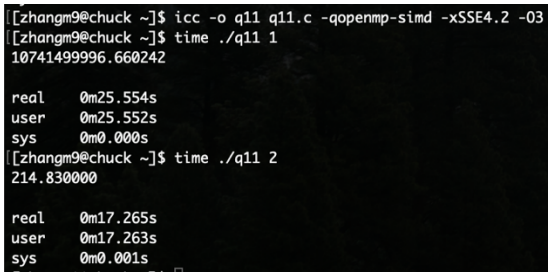
Switch back to the original memory setup (unaligned, dynamically allocated). Remove the literals (the 0.027s) from inside the two tight loops. What effect does this have on the timing? If there are any differences, can you use this to guess the cause of some of the performance differences between the two compilers?

```
/* This attribute asks the compiler not to inline the function  *
 * --- we don't want this adding extra optimisations here.      */
__attribute__ ((noinline))
double loops_v1(double *a, double *b, double *c, const int N, const int R)
{

    for (int r=0; r<R; r++)
    {
        #pragma omp simd
        for (int i=0; i<N; i++)
            a[i] += b[i] + c[i];
    }

    return a[N-1];
}
```

```
__attribute__ ((noinline))
double loops_v2(double *a, double *b, double *c,
                const int test_num, const int N, const int R)
{

    for (int r=0; r<R; r++)
    {
        #pragma omp simd
        for (int i=0; i<N; i++)
            a[i] = b[i] + c[i];

        /* To prevent the compiler from taking a McClane optimisation */
        if (a[N-1] < test_num-2)
            a[0] += 1.0;
    }

    return a[N-1];
}
```

GCC code	gcc -o q11 q11.c -fopenmp-simd -msse4.2 -O3
Before remove the literals (the 0.027s):	
Loop type	time ./q11 1
User time	0m25.537s
Return value	10741499996.660242
After remove the literals (the 0.027s):	
Loop type	time ./q11 2
User time	0m25.268s
Return value	214.830000





ICC code	icc -o q11 q11.c -qopenmp-simd -xSSE4.2 -O3
Before remove the literals (the 0.027s):	
Loop type	time ./q11 1
User time	0m25.552s
Return value	10741499996.660242
After remove the literals (the 0.027s):	
Loop type	time ./q11 2
User time	0m17.263s
Return value	214.830000