SKYLAKE AVX-512 SPECIFIC PATTERN OPTIMIZATIONS

https://godbolt.org/z/x7gNfb

Compress Loop Pattern

Auto-vectorization

```
int compress(double *a, double * __restrict b, int na)
{
    int nb = 0;
    for (int ia=0; ia <na; ia++)
    {
        if (a[ia] > 0.)
            b[nb++] = a[ia];

    }

    return nb;

}
```

VCOMPRESSPD|PS|D|Q	Store sparse packed floating-point values into dense memory
VEXPANDPD|PS|D|Q	Load sparse packed floating-point values from dense memory

double/single-precision/doubleword/quadword

```
vcompresspd YMMWORD PTR [rsi+rax*8]{k1}, ymm1
```

https://godbolt.org/z/x7gNfb

Compress Loop Pattern

Auto-vectorization

Targeting Intel® AVX2

-xcore-avx2 -qopt-report-file=stderr -qopt-report-phase=vec

LOOP BEGIN

remark #15344: loop was not vectorized: vector dependence prevents vectorization.

remark #15346: vector dependence: assumed FLOW dependence between b[nb] (7:4) and a[ia] (7:4)

LOOP END

Targeting Intel® AVX-512

-xcore-avx512 -qopt-report-file=stderr -qopt-report-phase=vec

LOOP BEGIN

remark #15300: LOOP WAS VECTORIZED

LOOP END

```
movsxd      rax, eax
xor         r11d, r11d
kmovw       r8d, k1
popcnt      r11d, r8d
vcompresspd YMMWORD PTR [rsi+rax*8]{k1}, ymm1
add         eax, r11d
```

Key Take Aways

Compress/Expand loop pattern doesn't vectorize on architectures like Intel® AVX2 and the previous ones and does with Intel® AVX-512

https://godbolt.org/z/x7gNfb

Compress Loop Pattern

Auto-vectorization

Let's try:

- Replace *-xcore-avx512* with *-xcascadelake*
- Set *-qopt-report=3* and check more detailed opt report
- Use *-qopt-zmm-usage=high* to target ZMM vector
- Remove *__restrict*
- gcc with *-mcascadelake*

https://godbolt.org/z/0kd2mT

Compress Loop Pattern

Complex example

```
int compress(int n1, int n2, float a[][n2], float b[__restrict])
{
  int nb = 0;
  for (int i1 = 0; i1 < n1; i1++)
  {
    float sc = 0.f;
    for (int i2 = 0; i2 < n2; i2++)
      sc += a[i1][i2];
    if (sc > 0.f)
      b[nb++] = sc;
  }
return nb;
}
```

https://godbolt.org/z/0kd2mT

Compress Loop Pattern

Complex example

remark #15541: outer loop was not auto-vectorized: consider using SIMD directive

```
#pragma omp simd
  for (int i1 = 0; i1 < n1; i1++)
  {
    float sc = 0.f;
    for (int i2 = 0; i2 < n2; i2++)
      sc += a[i1][i2];
#pragma omp ordered simd monotonic(nb:1)
    if (sc > 0.f)
      b[nb++] = sc;
  }
```

Key Take Aways

1. Outer loop vectorization can be achieved using OpenMP SIMD pragma.
2. The Compress/Expand loop pattern can hinted to compiler using monotonic clause.
3. The ordered clause takes into account the nb dependency (if omitted, wrong results)

https://godbolt.org/z/0kd2mT

Compress Loop Pattern

Complex example

Let's try:

- Add *simdlen(16)* to control ZMM registers usage
- Remove *simd monotonic* clause for the nested loop
- Remove all *simd* pragmas

Compress Loop Performance

Compiler Options	Speedup (in C)	Speedup (in FORTRAN)
Simple Loops (–O2 –xCORE-AVX2)	1.0x	1.0x
(-O2 –xCORE-AVX512)	12.8x	12.2x
Nested Loops (-O2 –xCORE-AVX512)	1.0x	1.0x
Ordered (-O2 –xCORE-AVX2 –qopenmp-simd)	1.8x	
Monotonic (-O2 –xCORE-AVX512 –qopenmp-simd)	7.3x	

Key Take Aways

1. Ordered clause will serialize the execution of the compress logic
2. Monotonic clause hints the compiler on the specific loop pattern and helps code generation in picking vcompress/vexpand vector instruction which leads to better performance.

Performance tests are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary.
The results above were obtained on an Intel® Xeon® Platinum 8168 system, frequency 2.7 GHz, running Red Hat* Enterprise Linux* Server 7.2 and using the Intel® Fortran Compiler version 18.0 update 1.

https://godbolt.org/z/pqjYPM

Histogram Loop pattern

Auto-vectorization

```
for (i=0; i<n; i++)
{
  y = sinf(x[i]*twopi);
  ih = floor((y-bot)*invbinw);
  ih = ih > 0 ? ih : 0;
  ih = ih < nbin ? ih : nbin;
  h[ih] = h[ih] + 1;
}
```

VPCONFLICT instruction detects elements with previous conflicts in a vector of indexes

- Allows to generate a mask with a subset of elements that are gurarnteed to be conflict free

```
vpconflictd zmm1{k1}, zmm2
```

Store to h is a scatter

ih can have the same value for different values of i

Vectorization with a SIMD directive would cause incorrect results

Histogram Loop pattern

https://godbolt.org/z/pqjYPM

Auto-vectorization

Targeting Intel® AVX2

-xcore-avx2 -qopt-report-file=stderr -qopt-report-phase=vec -qopt-report=3

LOOP BEGIN

remark #15344: loop was not vectorized: vector dependence prevents vectorization.

remark #15346: vector dependence: assumed FLOW dependence between h[ih] (13:5) and h[ih] (13:5)

LOOP END

Targeting Intel® AVX-512

-xcascadelake -qopt-report-file=stderr -qopt-report-phase=vec -qopt-report=3

```
vpminsd   ymm25, ymm4, ymm11                          #12.5
vpconflictd ymm5, ymm25                               #13.5
vpaddd    ymm26, ymm25, DWORD BCST .L_2il0floatpacket.7[rip]
                                                      #13.1
                                                      #13.5
```

LOOP BEGIN

remark #15300: LOOP WAS VEC

LOOP END

Key Take Aways

Histogram loop pattern doesn't vectorize on architectures like Intel® AVX2 and the previous ones and does with Intel® AVX512

Histogram Loop pattern

Complex example

https://godbolt.org/z/7nc8aP

```
for (int i=0; i<n; i++)
{
  float y = myfun(x[i]);
  int ih = floor( (y-bot)*invbinw );
  ih = ih >= 0 ? ih : 0;
  ih = ih <= nbin-1 ? ih : nbin-1;
  ++contents[ih];
}
```

remark #15543: loop was not vectorized: loop with function call not considered an optimization candidate.

Histogram Loop pattern

Complex example

https://godbolt.org/z/7nc8aP

Can be vectorized with OpenMP* by:

- Making myfun() a SIMD function
- Using the OMP ORDERED SIMD pragma/directive
- Add the OVERLAP hint to help compiler vectorize more efficiently

```
#pragma omp declare simd
float myfun(float x);

#pragma omp simd
for (int i=0; i<n; i++)
{
  float y = myfun(x[i]);
  int ih = floor( (y-bot)*invbinw );
  ih = ih >= 0 ? ih : 0;
  ih = ih <= nbin-1 ? ih : nbin-1;
#pragma omp ordered simd overlap(ih)
  ++contents[ih];
}
```

Key Take Aways

1. Outer loop vectorization can be achieved using OpenMP SIMD pragma.
2. The Histogram loop pattern can hinted to compiler using overlap clause.
3. The ordered clause takes into account the nb dependency (if omitted, wrong results)

Histogram Loop pattern

Complex example

https://godbolt.org/z/uR-eB8

Let's try:

- Remove *overlap* clause
- Remove *omp ordered pragma*
- Use *-qopt-zmm-usage=high* to target ZMM vector
- Check optimization report carefully. Any calls to action?
- Replace function declaration with implementation
  - add __attribute__((noinline))
  - anything new in the opt report?

Histogram Loop pattern

vecabi compiler option

https://godbolt.org/z/uR-eB8

```
#pragma omp declare simd
float myfun(float x) {
  float twopi=2.f*acosf(-1.f);
  float y = sinf(x*twopi);
  return y;
}
```

Compiler creates both vector and scalar versions

Use -vecabi=cmdtarget to target instruction set specified by –x switch

Else ABI requires arguments to be passed using xmm registers (Intel® SSE)

Linear(ref) clause avoids “gather” of vector of addresses

- Needed because Fortran default is pass by reference, not value

Key Take Aways

1. Outer loop vectorization can be achieved using OpenMP SIMD pragma.
2. The Histogram loop pattern can hinted to compiler using overlap clause.
3. The ordered clause takes into account the nb dependency (if omitted, wrong results)

Histogram Performance

Version of the program	Performance in seconds
Intel® AVX2 (non-vectorized)	59 seconds
Intel® AVX512 (vectorized)	6.6 seconds (~9x speedup)

Key Take Aways

Speedup really depends on the data set being histogramed. Lesser the conflict within the SIMD register, more is the speedup.

Performance tests are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary.
The results above were obtained on an Intel® Xeon® Platinum 8168 system, frequency 2.7 GHz, running Red Hat* Enterprise Linux* Server 7.2 and using the Intel® Fortran Compiler version 18.0 update 1.

Intel® AVX-512 generation for SKX

- Compile with processor-specific option *[-/Q]xCORE-AVX512*
- By default it will not optimize for more restrained ZMM register usage which works best for certain applications

A new compiler option *[-q/Q]opt-zmm-usage=low|high* is added to enable a smooth transition from AVX2 to AVX-512

```
void foo(double *a, double *b, int size) {
  #pragma omp simd
  for(int i=0; i<size; i++) {
     b[i]=exp(a[i]);
  }
}
```

```
icpc -c -xCORE-AVX512 -qopenmp -qopt-report:5 foo.cpp
```

```
remark #15305: vectorization support: vector length 4
  …
remark #15321: Compiler has chosen to target XMM/YMM
vector. Try using -qopt-zmm-usage=high to override
  …
remark #15478: estimated potential speedup: 5.260
```

https://tinyurl.com/tunesimd

Lab exercises

- 2 options:
  - Check generated ASM and opt report using Godbolt links

    Complete “let’s try” tasks
  - Try full version of examples from github on Intel DevCloud

    git clone https://github.com/fbaru-dev/hpc-workshop.git

Exercise 1 – skx_512

NB: Set *ulimit –s unlimited* before run

git clone https://github.com/fbaru-dev/hpc-workshop.git

You should observe the difference between the two cases and look at the assembly code

- Go to the folder skylake-avx512/compress/01
- Type *make* to compile the default case with AVX2. The compiler report is generated and you can read/interpret it. It does not vectorize the compress loop.
- Type *make run* to run the test and measure the timing.
- Type *make AVX512=yes* to compile for the AVX512, observe the change in the compiler report, run the test with *make run* and measure the timing.
- Generate the assembly code with *make AVX512 asm.*

Exercise 2 – skx_512

NB: Set *ulimit –s unlimited* before run

You should observe the difference between the two cases

- Go to the folder skylake-avx512/compress/02
- Type *make AVX512=yes* to compile the default case with AVX512. The compiler report is generated and you can read/interpret it. It vectorizes the inner loop.
- Type *make run* to run the test and measure the timing.
- Type *make AVX512=yes SIMD=yes* to compile with openmp simd enabled, observe the change in the compiler report, run the test with *make run* and measure the timing.

Exercise 3 – skx_512

NB: Set *ulimit –s unlimited* before run

You should observe the difference between the two cases and look at the assembly code

- Go to the folder skylake-avx512/histo/01
- Type *make* to compile the default case with AVX2. The compiler report is generated and you can read/interpret it. It does not vectorize the histogram patetrn loop.
- Type *make run* to run the test and measure the timing.
- Type *make AVX512=yes* to compile for the AVX512, observe the change in the compiler report, run the test with *make run* and measure the timing.
- Generate the assembly code with *make AVX512 asm.*

Exercise 4 – skx_512

NB: Set *ulimit –s unlimited* before run

You should observe the difference between the two cases and look at the simd vectorized code

- Go to the folder skylake-avx512/histo/02
- Type *make* to compile the default case with AVX512. The compiler report is generated and you can read/interpret it. It does not vectorize the histogram pattern loop.
- Type *make run* to run the test and measure the timing.
- Type *make SIMD=yes* to compile the SIMD version, observe the change in the compiler report, run the test with *make run* and measure the timing.

Legal Disclaimers and Optimization Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

All products, platforms, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel Performance Benchmark Limitations.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See www.intel.com/products/processor_number for details.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. § For more information go to www.intel.com/benchmarks.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804.

The Intel Core and Itanium processor families may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

The benchmark results reported may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

The code names Arrandale, Bloomfield, Boazman, Boulder Creek, Calpella, Chief River, Clarkdale, Cliffside, Cougar Point, Gulftown, Huron River, Ivy Bridge, Kilmer Peak, King's Creek, Lewisville, Lynnfield, Maho Bay, Montevina, Montevina Plus, Nehalem, Penryn, Puma Peak, Rainbow Peak, Sandy Bridge, Sugar Bay, Tylersburg, and Westmere presented in this document are only for use by Intel to identify a product, technology, or service in development, that has not been made commercially available to the public, i.e., announced, launched or shipped. It is not a "commercial" name for products or services and is not intended to function as a trademark.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel, Intel Core, Core Inside, Itanium, and the Intel Logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Intel collects and uses personal information from employees as part of SES, including capturing audio recording of sessions (both presenters and audience QA) as well as photographs and video recording of various event activities during the event. By registering and attending the SES conference, you give your consent for this capture. This includes both speakers and attendees. Intel will not retain your personal information longer than is necessary for the purposes for which it is collected.

egal Disclaimer & Optimization Notice

RMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO NTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND L DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR RANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, YRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ware and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. rmance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, ations and functions. Any change to any of those factors may cause the results to vary. You should consult other information