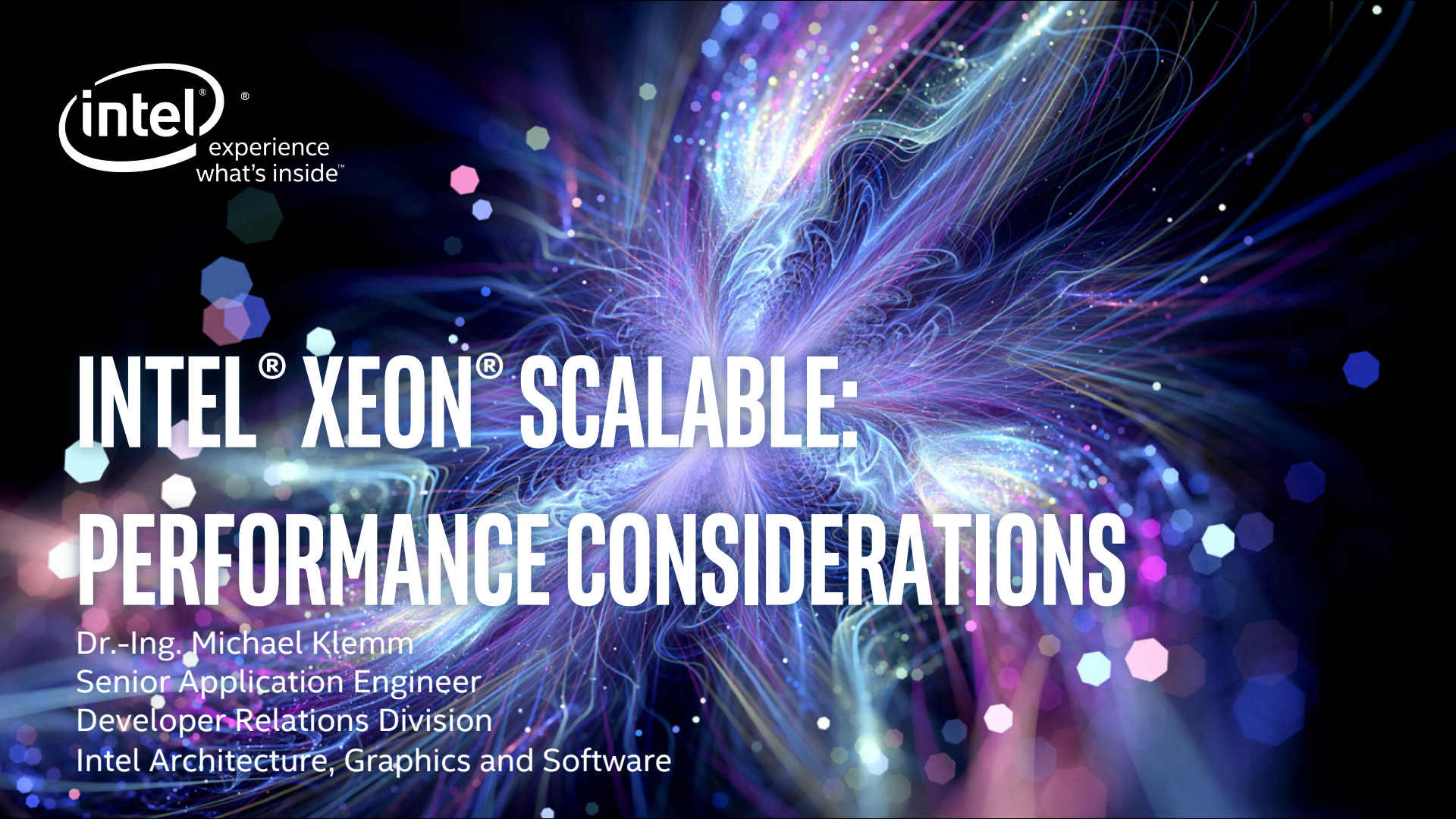


INTEL® XEON® SCALABLE: PERFORMANCE CONSIDERATIONS

Dr.-Ing. Michael Klemm
Senior Application Engineer
Developer Relations Division
Intel Architecture, Graphics and Software

Contents

- Application Performance Snapshot
- SIMD Programming
- Skylake Performance Considerations
- MPI Tuning

Notices and Disclaimers

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer. No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described 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.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, the Intel logo, Intel Optane and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the united states and other countries.


intel
experience
what's inside™
APPLICATION PERFORMANCE SNAPSHOT

Intel® Performance Snapshots

Three Fast Ways to Discover Untapped Performance

Is your application making good use of modern computer hardware?

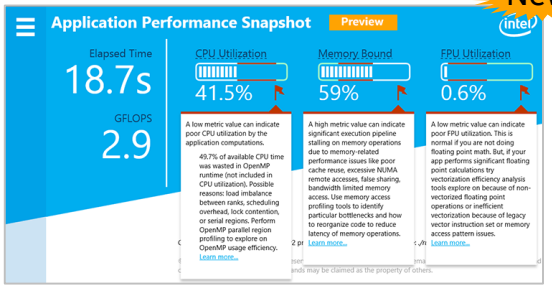
- Run a test case during your coffee break.
- High level summary shows which apps can benefit most from code modernization and faster storage.

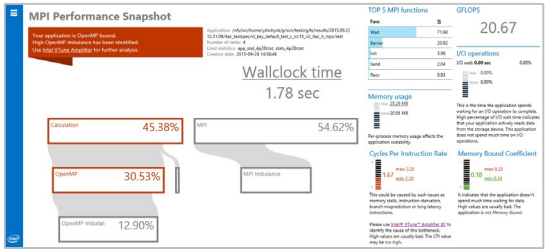
Pick a Performance Snapshot:

- **Application** – for non-MPI apps
- **MPI** – for MPI apps
- **Storage** – for systems. Servers and workstations with directly attached storage.

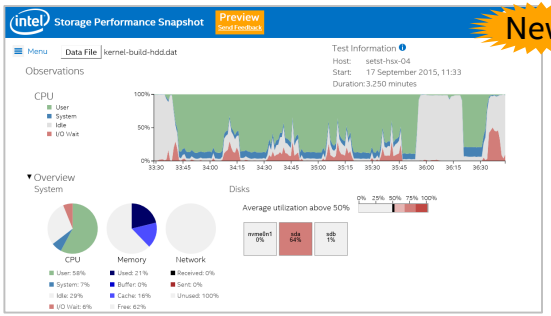
Free download: http://www.intel.com/performance-snapshot
Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.

New!





New!



Preview!

Application Performance Snapshot

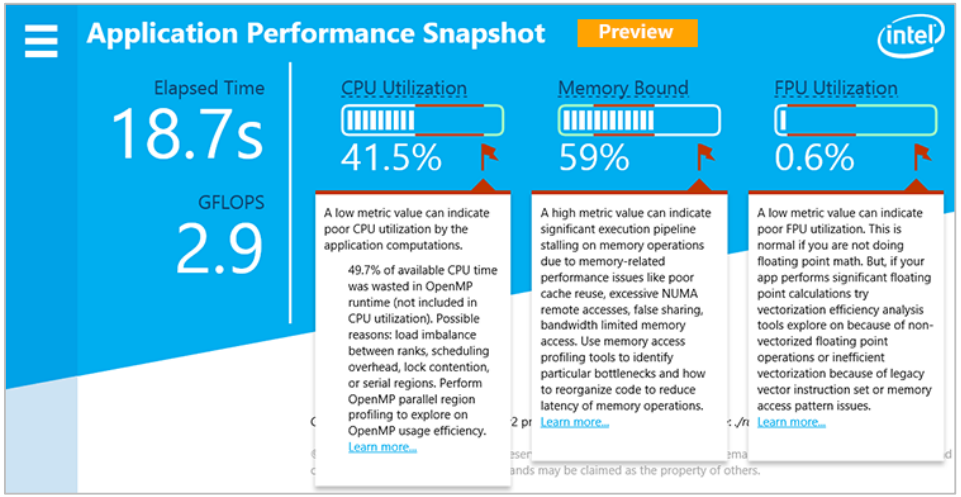
Discover opportunities for better performance with vectorization & threading

Objectives

- Simple enough to run during a coffee break
- Highlight where code modernization can help

Users

- Performance teams – fast prioritization of which apps will benefit most
- All Developers – size the potential performance gain from code modernization



Non-Objectives

- Actionable tuning data – that is another tool. Snapshot is just a fast “health” check.

Free download: http://www.intel.com/performance-snapshot

Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.

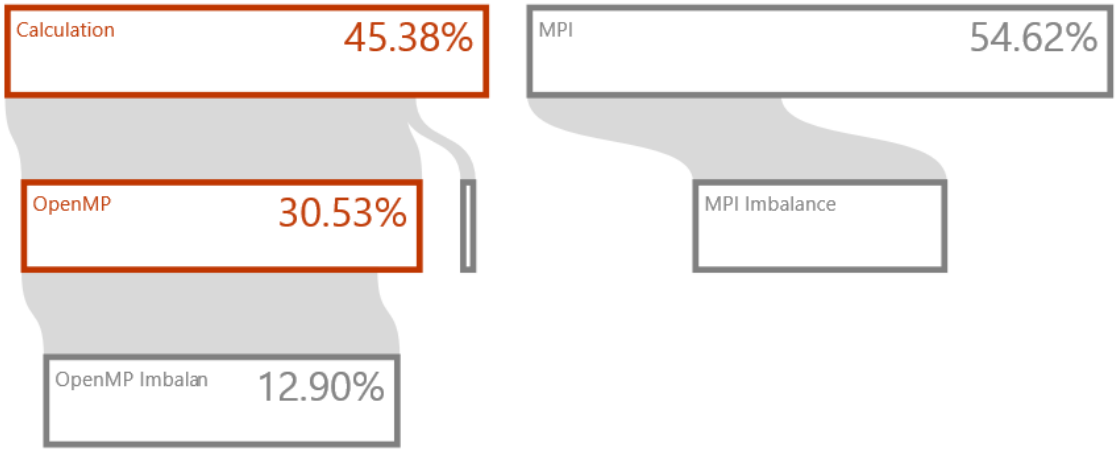
MPI Performance Snapshot

Your application is OpenMP bound.
High OpenMP imbalance has been identified.
Use Intel VTune Amplifier for further analysis.

Application: /nfs/inn/home/yshchyok/p/svn/testing/ts/results/2015.09.23
12.31.09/itac_testspec/vt_key_default_test_c_icc15_n2_itac_it_mps/test
Number of ranks: 4
Used statistics: app_stat_4p28t.txt, stats_4p28t.txt
Creation date: 2015-09-28 14:58:48

Wallclock time

1.78 sec



TOP 5 MPI functions

Func	%
Wait	71.98
Barrier	20.92
Init	3.98
Send	2.04
Recv	0.93

GFLOPS

20.67

I/O operations

I/O wait: **0.00 sec** 0.00%

PEAK 0.00%
MEAN 0.00%

This is the time the application spends waiting for an I/O operation to complete. High percentage of I/O wait time indicates that your application actively reads data from the storage device. This application does not spend much time on I/O operations.

Memory usage

PEAK 25.29 MB
MEAN 20.95 MB

Per-process memory usage affects the application scalability.

Cycles Per Instruction Rate

4
1.67
0
max: 2.23
min: 1.10

This could be caused by such issues as memory stalls, instruction starvation, branch misprediction or long latency instructions.

Please use Intel® VTune™ Amplifier XE to identify the cause of this bottleneck. High values are usually bad. The CPI value may be *too high*.

Memory Bound Coefficient

1
0.18
0
max: 0.23
min: 0.14

It indicates that the application doesn't spend much time waiting for data. High values are usually bad. The application is *not Memory Bound*.

Free download: http://www.intel.com/performance-snapshot. Also included with Intel® Parallel Studio Cluster Edition.

Storage Performance Snapshot

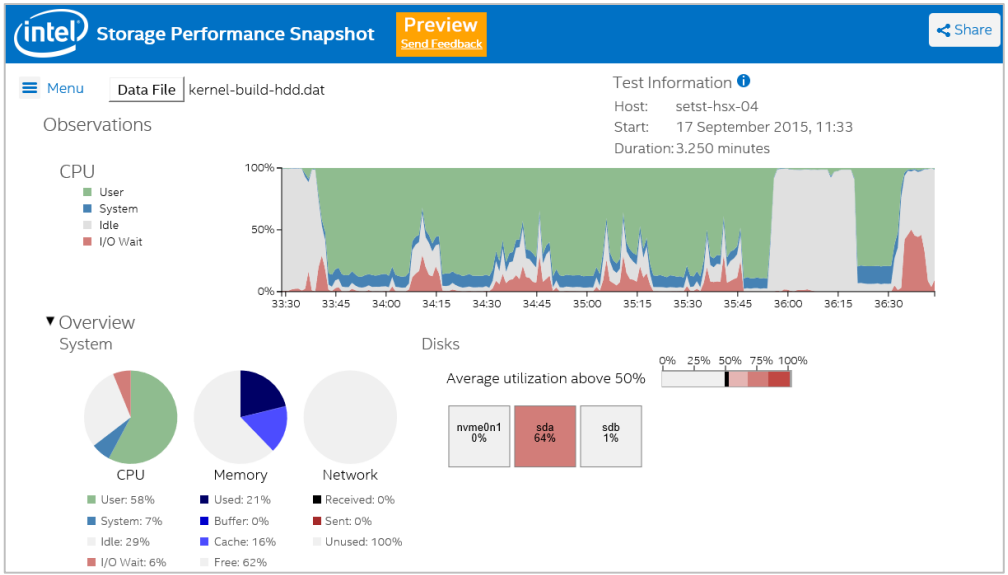
Discover if faster storage can improve server/workstation performance

Learn It On One Coffee Break

- Easy setup
- Quickly see meaningful data
- System view of workload
- Any architecture

Targeted Systems

- Servers & workstations with directly attached storage
- Not scale out storage clusters
- Linux kernel 2.6 or newer dstat 0.7 or newer
- Windows Server 2012, Windows 8 or newer Windows OS



Free download: http://www.intel.com/performance-snapshot

Also included with Intel® Parallel Studio and Intel® VTune™ Amplifier products.

APS Usage

Setup Environment

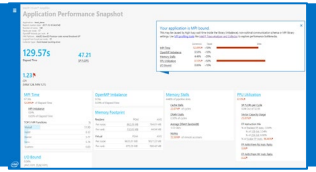
- source <APS_Install_dir>/apsvars.sh

Run Application

- mpirun <mpi options> aps <application and args>

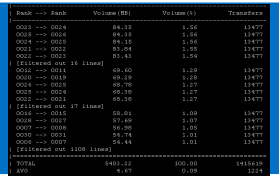
Generate Report on Results

- aps -report <result folder>



Generate advanced CL reports on Results

- aps-report -<option> <result folder>

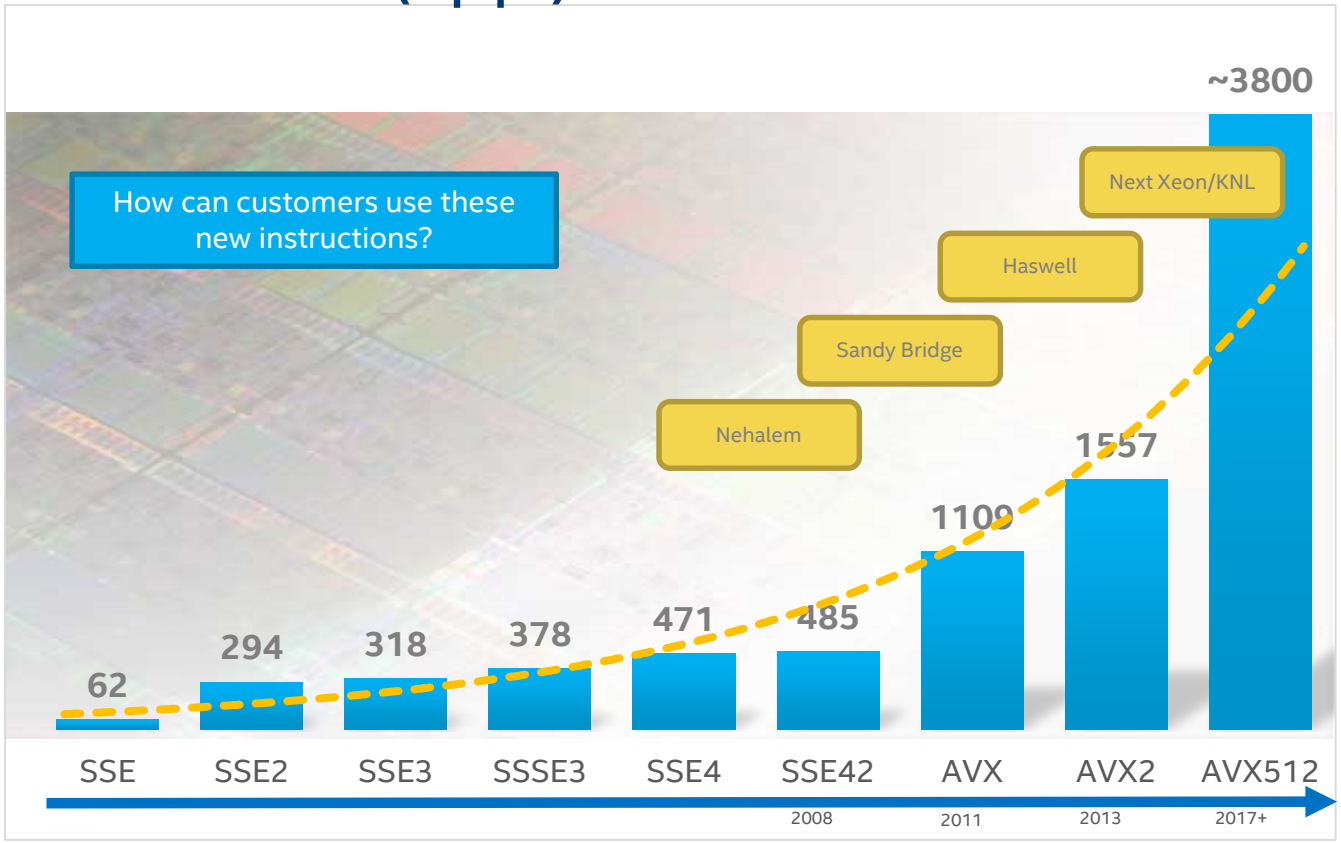


intel®
experience
what's inside™

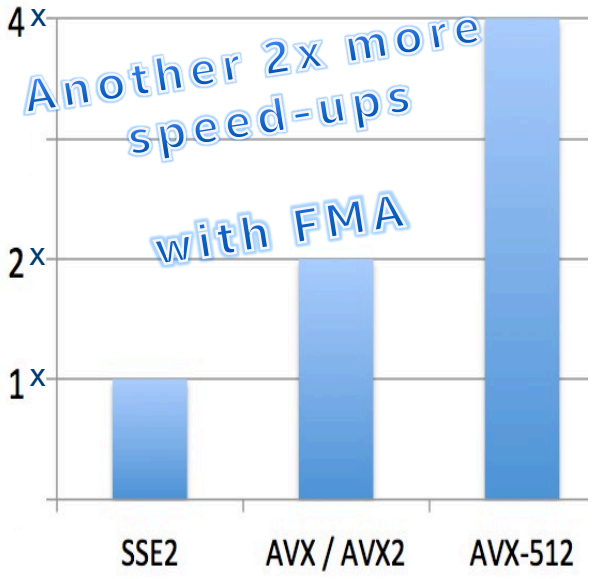
VECTOR PARALLELISM: WHY DO WE CARE?

Vector
Processing

Cumulative (app.) # of Vector Instructions



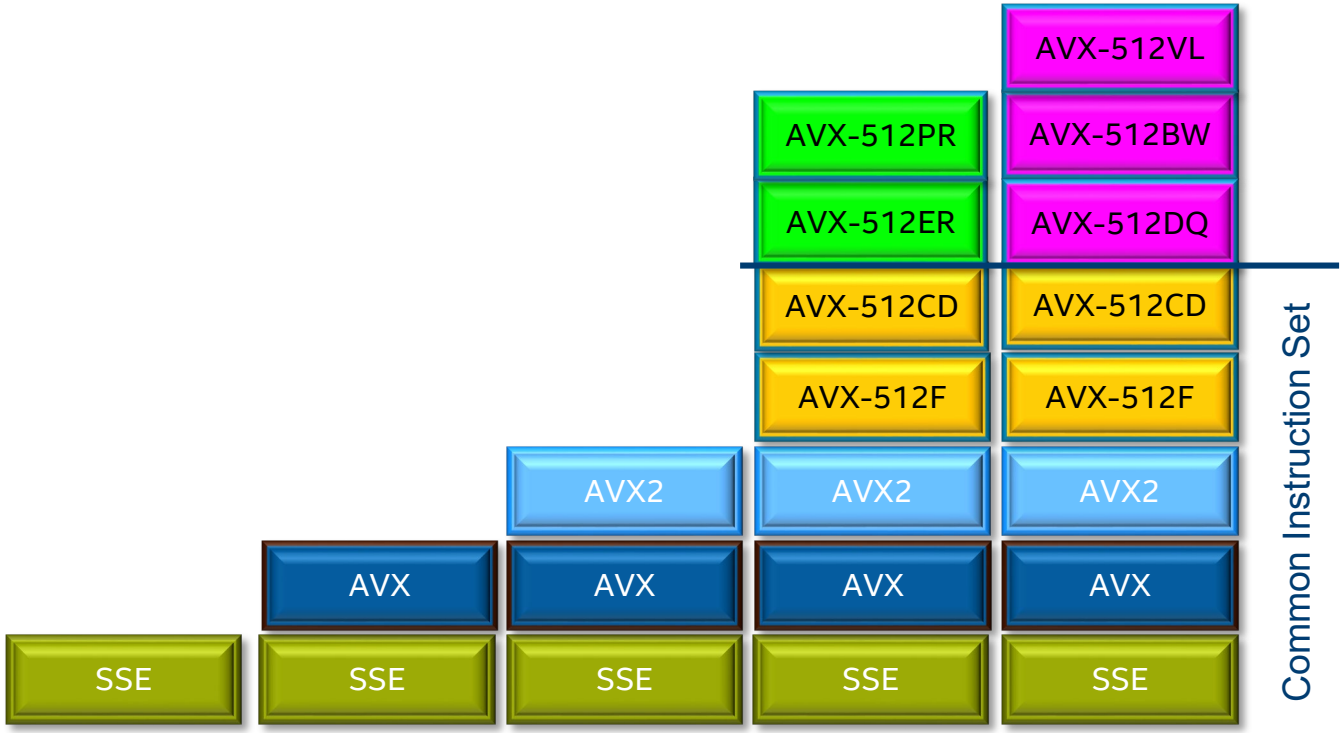
16x DP speed-up over scalar.
8x DP speed-up over SSE
with Advanced Vector Extensions 512 (AVX-512)



- Significant leap to 512-bit SIMD support for processors
- Intel® Compilers and Intel® Math Kernel Library include AVX-512 support
- Strong compatibility with AVX
- Added EVEX prefix enables additional functionality
- Appears first in future Intel® Xeon Phi™ coprocessor, code named Knights Landing

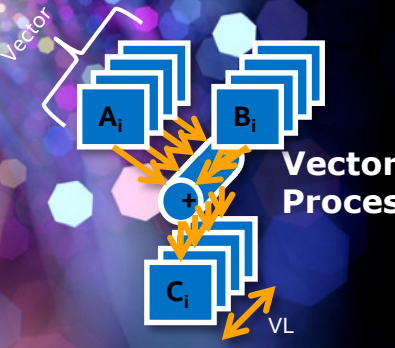
Higher performance for the most demanding computational tasks

Intel® AVX-512 - Comparison

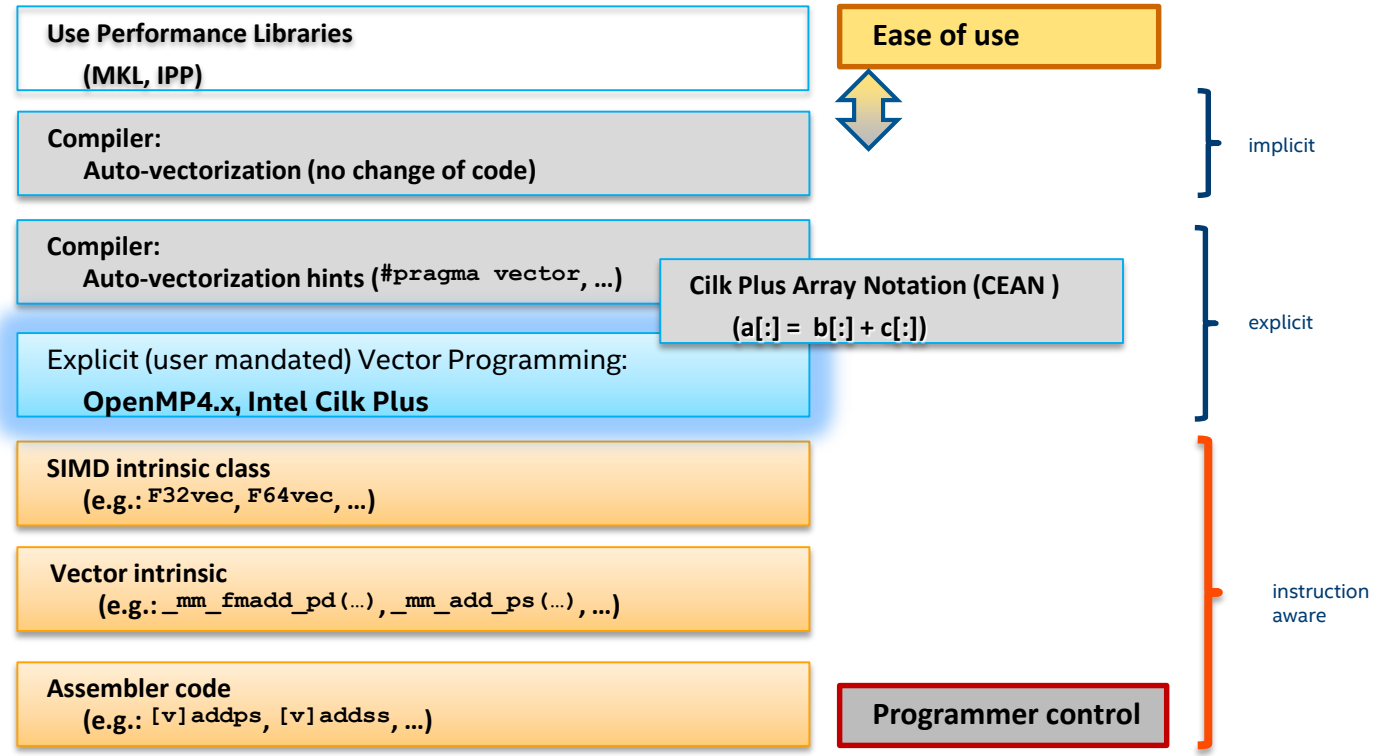


intel® experience what's inside™

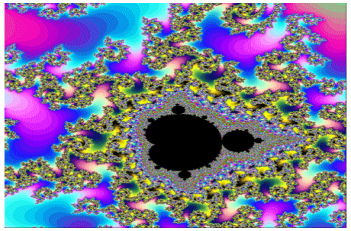
SIMD AND OPENMP* SIMD

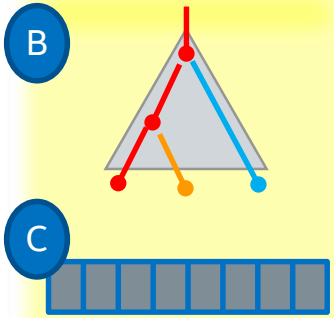


Many Ways to Vectorize



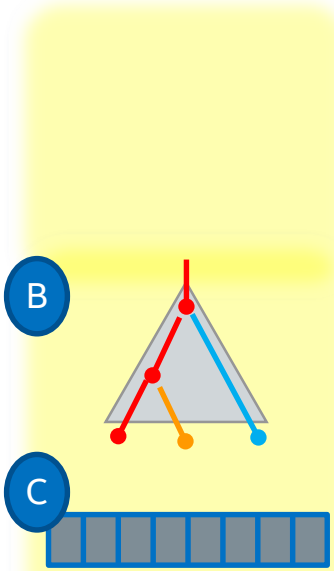
2 level parallelism decomposition with **OpenMP4.x**: image processing example

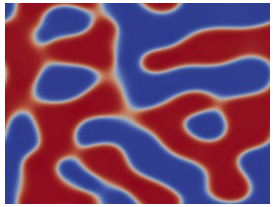





```
#pragma omp parallel for
for (int y = 0; y < ImageHeight; ++y){
#pragma omp simd
   for (int x = 0; x < ImageWidth; ++x){
      count[y][x] = mandel(in_vals[y][x]);
   }
}
```

2L parallelism decomposition with **OpenMP4.x**: fluid dynamics example





```
#pragma omp parallel for
for (int i = 0; i < X_Dim; ++i){
#pragma omp simd
   for (int m = 0; x < n_velocities; ++m){
      next_i = f(i, velocities(m));
      X[i] = next_i;
   }
}
```

Factors that **prevent Vectorizing** your code

1. Loop-carried dependencies

```
DO I = 1, N
   A(I + M) = A(I) + B(I)
ENDDO
```

1.A Pointer aliasing (compiler-specific)

```
void scale(int *a, int *b)
{
   for (int i = 0; i < 1000; i++)
         b[i] = z * a[i];
}
```

2. Function calls (incl. indirect)

```
for (i = 1; i < nx; i++) {
  x = x0 + i * h;
  sumx = sumx + func(x, y, xp);
}
```

3. Loop structure, boundary condition

```
struct _x { int d; int bound; };

void doit(int *a, struct _x *x)
{
  for(int i = 0; i < x->bound; i++)
    a[i] = 0;
}
```

4 Outer vs. inner loops

```
for(i = 0; i <= MAX; i++) {
  for(j = 0; j <= MAX; j++) {
    D[j][i] += 1;
  }
}
```

5. Cost-benefit (compiler specific..)

And others……

Factors that **slow-down** your **Vectorized** code

1.A. Indirect memory access

```
for (i=0; i<N; i++)
      A[B[i]] = C[i]*D[i]
```

1.B Memory sub-system Latency / Throughput

```
void scale(int *a, int *b)
{
   for (int i = 0; i < VERY_BIG; i++)
         c[i] = z * a[i][j];
         b[i] = z * a[i];
}
```

2. Serialized or “sub-optimal” function calls

```
for (i = 1; i < nx; i++) {
         sumx = sumx +
         serialized_func_call(x,
y, xp);
}
```

3. Small trip counts not multiple of VL

```
void doit(int *a, int *b, int
unknown_small_value)
{
  for(int i = 0; i <
unknown_small_value; i++)
      a[i] = z*b[i];
}
```

4. Branchy codes, *outer vs. inner loops*

```
for(i = 0; i <= MAX; i++) {
         if ( D[i] < N)
                  do_this(D);
         else if (D[i] > M)
                  do_that();
         //…
 }
```

5. MANY others: spill/fill, fp accuracy trade-offs, FMA, DIV/SQRT, Unrolling, even AVX throttling..

Auto-vectorization

Auto vectorization only helps in some cases

- Increased complexity of instructions makes it hard for the compiler to select proper instructions
- Code pattern needs to be recognized by the compiler
- Precision requirements often inhibit SIMD code gen

Example: Intel® Composer XE

`-vec` (automatically enabled with –O3)

`-qopt-report`

`-qopt-report-file=[filename|stdout|stderr]`

`-qopt-report-annotate[=[text|html]]`

HTML Optimization Report

```
11      void simd_daxpy(double *x, double *y, double a, size_t n) {
```

```
INLINE REPORT: (simd_daxpy(double *, double *, double, size_t)) [2] /nfs/home/mklemm/tmp/frequency.c(11,59)

/nfs/home/mklemm/tmp/frequency.c(11,59):remark #34051: REGISTER ALLOCATION : [simd_daxpy] /nfs/home/mklemm/tmp/frequency.c:11

    Hardware registers
        Reserved     :    2[ rsp rip]
        Available    :   39[ rax rdx rcx rbx rbp rsi rdi r8-r15 mm0-mm7 zmm0-zmm15]
        Callee-save  :    6[ rbx rbp r12-r15]
        Assigned     :    8[ rax rdx rcx rsi rdi zmm0-zmm2]

    Routine temporaries
        Total         :      35
            Global    :      13
            Local     :      22
        Regenerable   :       4
        Spilled       :       0

    Routine stack
        Variables     :       0 bytes*
            Reads     :       0 [0.00e+00 ~ 0.0%]
            Writes    :       0 [0.00e+00 ~ 0.0%]
        Spills        :       0 bytes*
            Reads     :       0 [0.00e+00 ~ 0.0%]
            Writes    :       0 [0.00e+00 ~ 0.0%]

    Notes

        *Non-overlapping variables and spills may share s
         so the total stack size might be less than this.
```

```
28          for (size_t rep = 0; rep < OUTER_REP; ++rep) {
```

```
LOOP BEGIN at /nfs/home/mklemm/tmp/frequency.c(28,5)
   remark #15344: loop was not vectorized: vector dependence prevents vectorization. First dependence is shown below. Use level 5 report for details
   remark #15346: vector dependence: assumed OUTPUT dependence between call:simd_daxpy(double *, double *, double, size_t) (30:12) and call:scalar_d

   LOOP BEGIN at /nfs/home/mklemm/tmp/frequency.c(29,9)
      remark #15344: loop was not vectorized: vector dependence prevents vectorization. First dependence is shown below. Use level 5 report for deta
   LOOP END

   LOOP BEGIN at /nfs/home/mklemm/tmp/frequency.c(32,9)
      remark #15344: loop was not vectorized: vector dependence prevents vectorization. First dependence is shown below. Use level 5 report for deta
   LOOP END
LOOP END
```

```
12      #pragma vector always
13      #pragma ivdep
14          for (size_t i = 0; i < n; ++i) {
```

```
LOOP BEGIN at /nfs/home/mklemm/tmp/frequency.c(14,5)
<Peeled loop for vectorization>
LOOP END

LOOP BEGIN at /nfs/home/mklemm/tmp/frequency.c(14,5)
   remark #15300: LOOP WAS VECTORIZED
LOOP END
```

```
29              for (size_t in = 0; in < INNER_REP; ++in) {
30                 simd_daxpy(x, y, 5.0, ARRAYSZ);
31              }
32              for (size_t in = 0; in < INNER_REP; ++in) {
33                  scalar_daxpy(x, y, 5.0, ARRAYSZ);
34              }
35          }
36
37          return 0;
38      }
```

OpenMP SIMD Loop Construct

Vectorize a loop nest

- Cut loop into chunks that fit a SIMD vector register
- No parallelization of the loop body

Syntax (C/C++)

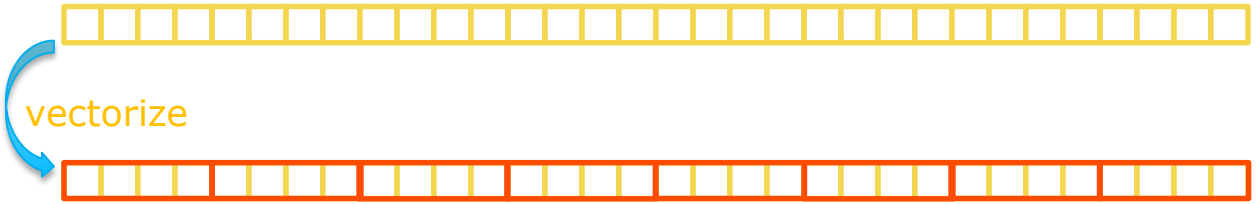
```
#pragma omp simd [clause[[,] clause],…]
for-loops
```

Syntax (Fortran)

```
!$omp simd [clause[[,] clause],…]
do-loops
```

Example

```
void sprod(float *a, float *b, int n) {
    float sum = 0.0f;
#pragma omp simd reduction(+:sum)
    for (int k=0; k<n; k++)
        sum += a[k] * b[k];
    return sum;
}
```



Data Sharing Clauses

`private(var-list):`
Uninitialized vectors for variables in *var-list*

`firstprivate(var-list):`
Initialized vectors for variables in *var-list*

`reduction(op:var-list):`
Create private variables for *var-list* and apply reduction operator *op* at the end of the construct

SIMD Loop Clauses

`safelen (length)`

- Maximum number of iterations that can run concurrently without breaking a dependence
- In practice, maximum vector length

`linear (list[:linear-step])`

- The variable's value is in relationship with the iteration number
  - $x_i = x_{orig} + i * \text{linear-step}$

`aligned (list[:alignment])`

- Specifies that the list items have a given alignment
- Default is alignment for the architecture

`collapse (n)`

SIMD Worksharing Construct

Parallelize and vectorize a loop nest

- Distribute a loop's iteration space across a thread team
- Subdivide loop chunks to fit a SIMD vector register

Syntax (C/C++)

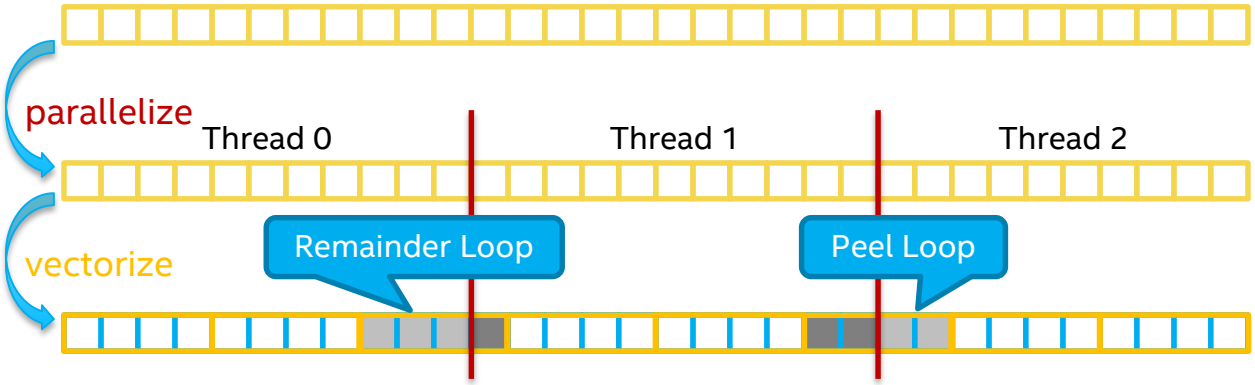
```
#pragma omp for simd [clause[[,] clause],…]
for-loops
```

Syntax (Fortran)

```
!$omp do simd [clause[[,] clause],…]
do-loops
[!$omp end do simd [nowait]]
```

Example

```
void sprod(float *a, float *b, int n) {
    float sum = 0.0f;
#pragma omp for simd reduction(+:sum)
    for (int k=0; k<n; k++)
        sum += a[k] * b[k];
    return sum;
}
```



SIMD Function Vectorization

Declare one or more functions to be compiled for calls from a SIMD-parallel loop

Syntax (C/C++):

```
#pragma omp declare simd [clause[[,] clause],…]
[#pragma omp declare simd [clause[[,] clause],…]]
[…]
function-definition-or-declaration
```

Syntax (Fortran):

```
!$omp declare simd (proc-name-list)
```

SIMD Function Vectorization

```
#pragma omp declare simd
float min(float a, float b) {
    return a < b ? a : b;
}

#pragma omp declare simd
float distsq(float x, float y) {
    return (x - y) * (x - y);
}

void example() {
#pragma omp parallel for simd
    for (i=0; i<N; i++) {
        d[i] = min(distsq(a[i], b[i]), c[i]);
}   }
```

```
_ZGVZN16vv_min(%zmm0, %zmm1):
    vminps %zmm1, %zmm0, %zmm0
    ret
```

```
_ZGVZN16vv_distsq(%zmm0, %zmm1):
    vsubps %zmm0, %zmm1, %zmm2
    vmulps %zmm2, %zmm2, %zmm0
    ret
```

```
vmovups (%r14,%r12,4), %zmm0
vmovups (%r13,%r12,4), %zmm1
call _ZGVZN16vv_distsq
vmovups (%rbx,%r12,4), %zmm1
call _ZGVZN16vv_min
```

AT&T syntax: destination operand is on the right

SIMD Function Vectorization

`simdlen (`*`length`*`)`

- generate function to support a given vector length

`uniform (`*`argument-list`*`)`

- argument has a constant value between the iterations of a given loop

`inbranch`

- optimize for function always called from inside an if statement

`notinbranch`

- function never called from inside an if statement

`linear (`*`argument-list[:linear-step]`*`)`

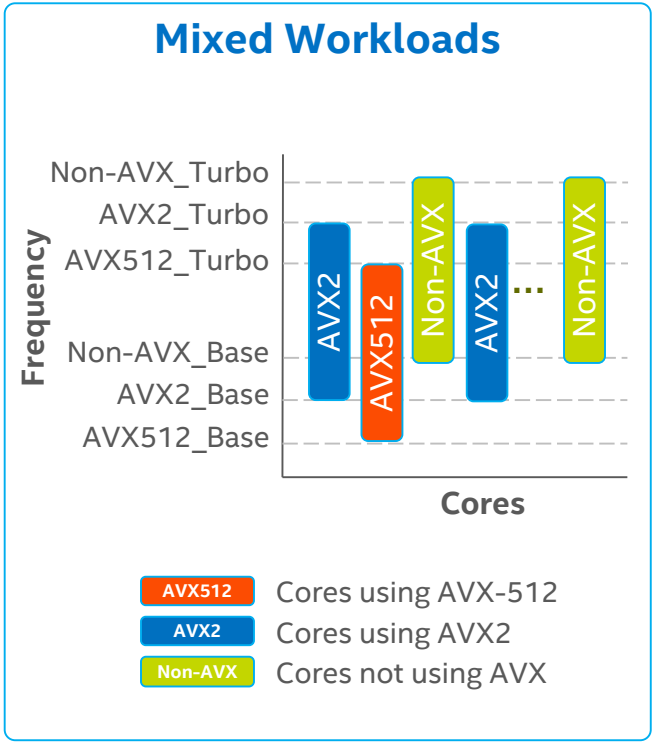
`aligned (`*`argument-list[:alignment]`*`)`


intel
experience
what's inside™
PERFORMANCE CONSIDERATIONS

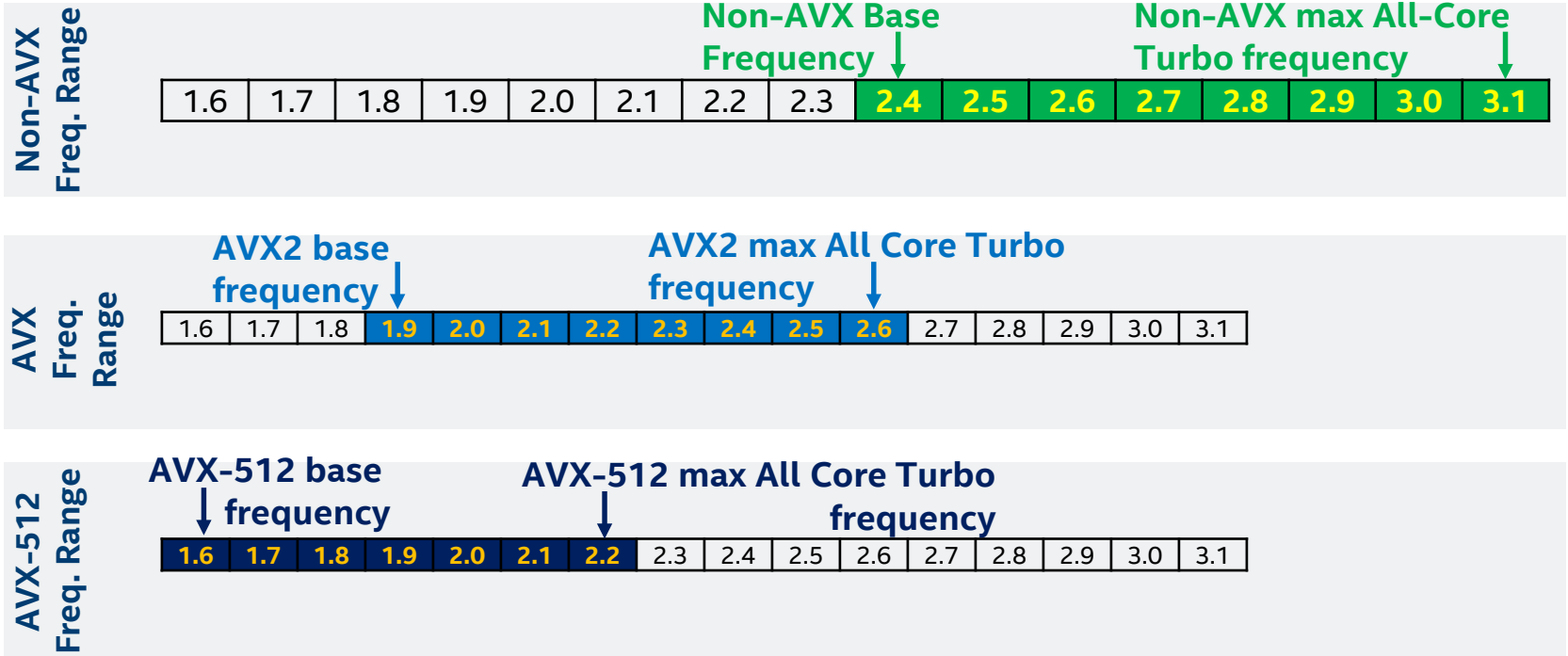
Frequency Behavior While Running Intel® AVX Code

- Cores running non-AVX, Intel® AVX2 light/heavy, and Intel® AVX-512 light/heavy code have different turbo frequency limits
- Frequency of each core is determined independently based on workload demand

Code Type	All Core Frequency Limit
SSE AVX2-Light (without FP & int-mul)	Non-AVX All Core Turbo
AVX2-Heavy (FP & int-mul) AVX512-Light (without FP & int-mul)	AVX2 All Core Turbo
AVX512-Heavy (FP & int-mul)	AVX512 All Core Turbo



AVX Frequency – All Core Turbo (SKX 6148)




intel
experience
what's inside™
MPI TUNING BASICS

Prerequisite –

Make sure your cluster is properly configured

- Install the latest Intel® MPI Library
- Have the documentation at hand
- When benchmarking try to run on the same subset of nodes
- Understand the performance characteristics of your cluster
  - What fabrics are available, how do they perform
    - What is the lowest achievable latency
    - What is the maximum achievable bandwidth
  - Communication speeds differ
    - Intra-socket / Inter-socket
    - Intra-node / Inter-node
  - Use IMB to determine performance

Prerequisite –

Understand the performance characteristics of your Application

- Simple code internal timings
- ITAC traces – use MPI_Pcontrol() to manage overhead
- IMPI statistics
- MPS statistics
- VTune traces

-> know what MPI routines to tune for

Use lightweight statistics

- Set **I_MPI_STATS** to a non-zero integer value to gather MPI communication statistics (max value is 10)
- Set **I_MPI_STATS=ipm** to get statistics in IPM format
- Default names are **stats.txt** or **stats.ipm** and can be changed setting **I_MPI_STATS_FILE** variable
- Change scope with **I_MPI_STATS_SCOPE** to increase effectiveness of the analysis
- Example here: Gromacs rank 0 with

```
I_MPI_STATS=3
I_MPI_STATS_SCOPE=coll
```

```
Communication Activity by actual args
Collectives
Operation       Context  Algo  Comm size    Message size      Calls  Cost(%)
-----------------------------------------------------------------------------
Allreduce
1                    58     1          4              24          1     0.00
2                    58     1          4               4          8     0.00
3                    58     1          4               8         12     0.03
4                    58     1          4            1376        181     0.04
5                    58     1          4            1344         19     0.01
6                    58     1          4            1216          1     0.00
7                    58     1          4             224          1     0.00
8                     0     5        192               8          2     0.00
9                     0     5        192             968          1     0.00
10                    0     5        192             288          2     0.01
11                    0     5        192             768          2     0.00
Barrier
1                    62     5        160               0          1     0.00
2                     0     5        192               0          1     0.00
Bcast
...
Gather
1                    52     3          5              32         25     0.01
2                    54     3          4              36         25     0.00
3                    56     3          8              28         25     0.01
Reduce
1                    60     1         40              24          1     0.00
2                    60     1         40               4          8     0.00
3                    60     1         40               8         12     0.01
4                    60     1         40            1376        181     0.21
5                    60     1         40            1344         19     0.03
6                    60     1         40            1216          1     0.00
7                    60     1         40             224          1     0.00
Scatter
1                    62     1        160               8          1     0.00
Scatterv
1                    62     1        160          315840          2     0.03
2                    62     1        160           52640          1     0.08
=============================================================================
```

Use best available communication fabric

Supported I_MPI_FABRICS	Description
shm	Shared-memory only; intra-node default
tcp	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand* (through IPoIB*)
~~dapl~~	~~DAPL-capable network fabrics, such as InfiniBand*, iWarp*, and XPMEM* (through DAPL*)~~
~~ofa~~	~~OFA-capable network fabric including InfiniBand* (through OFED* verbs)~~
tmi	Tag Matching Interface (TMI) capable network fabrics, such as Intel® True Scale Fabric, Intel® Omni-Path Architecture and Myrinet*
ofi	OFI (OpenFabrics Interfaces*)-capable network fabric including Intel® True Scale Fabric, and TCP (through OFI* API)

- Intel® MPI Library will select the fastest available fabric by default: shared memory within a node and Intel® Omni-Path Architecture (shm:tmi) or Infiniband* (shm:dapl) across nodes
- For the OpenFabrics Enterprise Distribution (OFED*) software stack, select shm:ofa
- Check the fabric being used by setting I_MPI_DEBUG to 2

Disable fallback for benchmarking

- Intel® MPI Library falls back from the 'tmi' or 'shm:tmi' fabric to 'tcp' and/or 'shm:tcp' if the TMI provider initialization failed
- Disable I_MPI_FALLBACK to ensure that the fast default fabric is working. Fallback is also disabled by default if I_MPI_FABRICS is set
- On the command line you can use:

```
$ mpirun -genv I_MPI_FABRICS tmi -genv I_MPI_FALLBACK 0 -n <nprocs> ./app
```

  or, equivalently, use the command line option -PSM2:

```
$ mpirun -PSM2 -n <nprocs> ./app
```

- Using these settings the run will fail if the high performance fabric is not available, rather than run at a degraded performance over a backup fabric.

Use connectionless communication

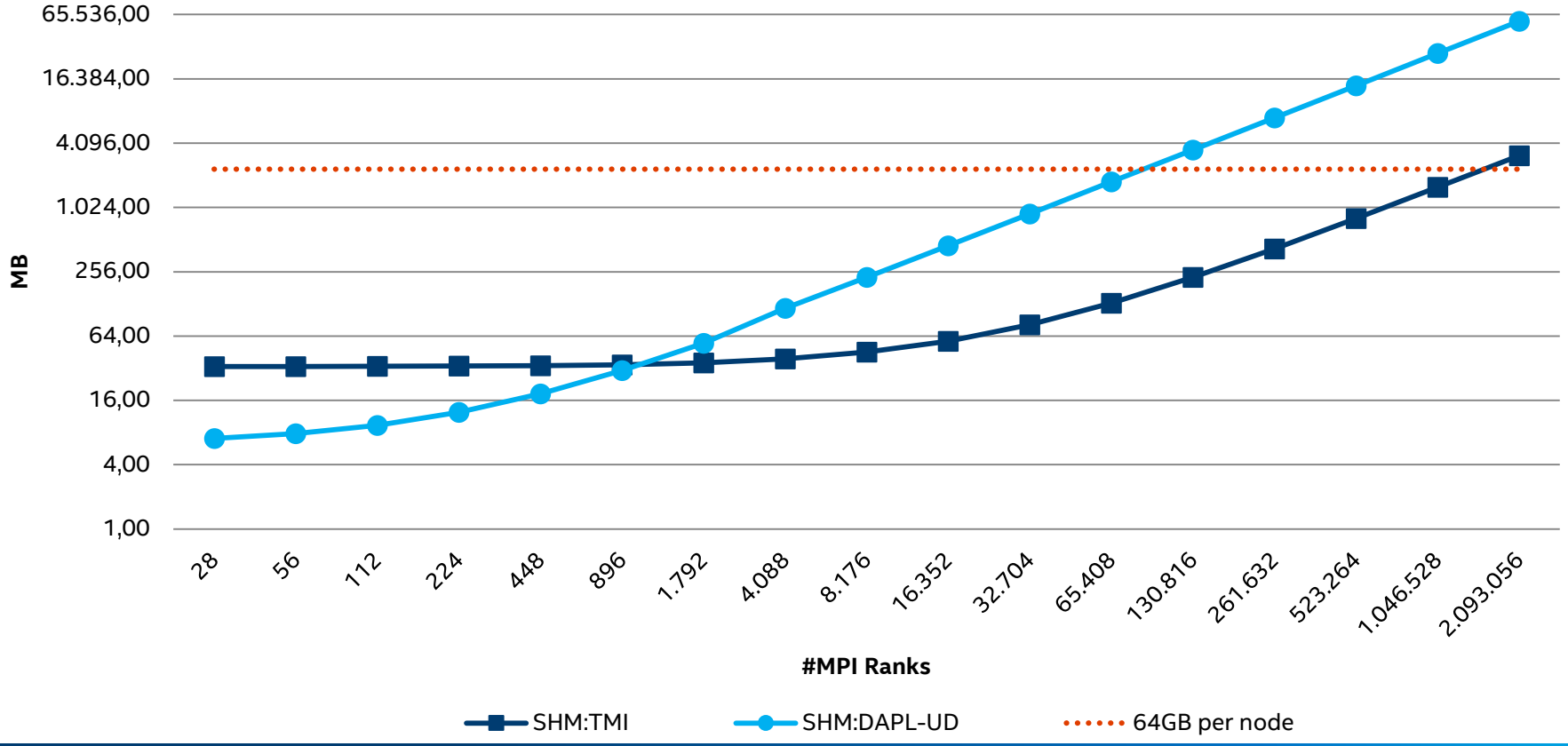
Intel® Omni-Path Architecture fabrics TMI and OFI automatically use the PSM2 provided connectionless communication features

For Infiniband* fabrics using a DAPL provider version higher than 2.0.27 this can be enabled via environmental variables

- Provides better scalability
- Reduces memory footprint by reducing the number of receive queues
- General use for large and memory hungry jobs > 1k MPI ranks recommended
- Automatically triggered for large jobs via I_MPI_LARGE_SCALE_THRESHOLD which is currently 4096 ranks

```
I_MPI_FABRICS=shm:dapl
I_MPI_DAPL_UD=1
I_MPI_DAPL_UD_PROVIDER=<chosen from /etc/dat.conf entries>
```

Intel MPI Library memory consumption model per rank. Alltoall connections. Worst case. Lower is better



Select a proper process pinning 1/3

- The default pinning is suitable for most scenarios
- Set I_MPI_PERHOST or use the –perhost (/-ppn) option to override the default process layout:

```
$ mpirun –ppn <#processes per node> -n <#processes> …
```

- Intel® MPI Library respects the batch scheduler settings - to overwrite use:

```
I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0
```

- Per-node pinning can also be achieved using a “machinefile”
- Custom processor core pinning can be achieved by two different environment variables

`I_MPI_PIN_PROCESSOR_LIST` - for pure MPI applications

`I_MPI_PIN_DOMAIN` - for Hybrid – MPI + Threading applications

Select a proper process pinning 2/3

- The '**`cpuinfo`**' utility from Intel MPI Library can be used to observe the processor topology
- Threads of Hybrid applications are not pinned by default
- Threads can migrate along the cores of a rank defined by **`I_MPI_PIN_DOMAIN`**
- Therefore, threads should be pinned as well using e.g. **`OMP_PLACES`**
- Further information can be found in the "Process Pinning" section of the Intel MPI Library reference manual

Select a proper process pinning 3/3

Default Intel Library MPI pinning	Impact
I_MPI_PIN=on	Pinning Enabled
I_MPI_PIN_MODE=pm	Use Hydra for Pinning
I_MPI_PIN_RESPECT_CPUSET=on	Respect process affinity mask
I_MPI_PIN_RESPECT_HCA=on	Pin according to HCA socket
I_MPI_PIN_CELL=unit	Pin on all logical cores
I_MPI_PIN_DOMAIN=auto:compact	Pin size #lcores/#ranks : compact
I_MPI_PIN_ORDER=compact	Order domains adjacent


intel
experience
what's inside™
MPI AUTO-TUNING

Cluster Specific Tuning with mpitune

Find optimal values for library tuning knobs on the particular cluster or application environment.

- Run it once after installation and each time after a cluster configuration change
- Best configuration is recorded for each combination of communication device, number of nodes, MPI ranks and the process distribution model
- Configuration is stored in Intel ® MPI folders and available to all users

```
# Collect configuration values:
$ mpitune [options]
```

```
# Reuse recorded values:
$ mpirun –tune ./application
```

Application Specific Tuning with mpitune

Find optimal values for library tuning knobs on the particular cluster or application environment.

- Run it for each application and after application- or cluster configuration change
- Best configuration is recorded for each combination of communication device, number of nodes, MPI ranks and the process distribution model
- Configuration is stored in user's home

```
# Collect configuration values:
$ mpitune [options]   \
--application \"mpirun application\"
```

```
# Reuse recorded values:
$ mpirun –tune ./app.conf \
./application
```

What is the actual search space of mpitune?

I_MPI_DYNAMIC_CONNECTION	I_MPI_ADJUST_GATHERV
I_MPI_RDMA_SCALABLE_PROGRESS	I_MPI_ADJUST_ALLGATHER
I_MPI_RDMA_TRANSLATION_CACHE	I_MPI_ADJUST_ALLGATHERV
I_MPI_WAIT_MODE	I_MPI_ADJUST_SCATTER
I_MPI_ADJUST_BCAST	I_MPI_ADJUST_SCATTERV
I_MPI_ADJUST_BARRIER	I_MPI_ADJUST_REDUCE
I_MPI_EAGER_THRESHOLD	I_MPI_ADJUST_ALLREDUCE
I_MPI_INTRANODE_EAGER_THRESHOLD	I_MPI_ADJUST_REDUCE_SCATTER
I_MPI_RDMA_EAGER_THRESHOLD	I_MPI_ADJUST_ALLTOALL
I_MPI_ADJUST_GATHER	I_MPI_ADJUST_ALLTOALLV

… for different fabrics, iterations, contents, ranks per node and overall rank counts …

Estimating the runtime of mpitune

The overall runtime of mpitune can be estimated by.:

#options x #option parameters x #iterations x application runtime

Can be determined using the scheduler only –so option of mpitune

Can be taken from the options.xml in $I_MPI_ROOT/etc64 but is quite complex – take the average of 5

Default is 3 iterations unless specified differently via the –I parameter

Should be known by the user / alternatively time the execution of the scheduler only mode

mpitune takes time … ! Faster?
-----*fast*-----

Leverage fast tuning to adjust the collective algorithms used by your application.

- mpitune --fast 1 – uses a separate mpitune binary -> different parameters
- Requires certain amount of time spent in collectives (10% default)
- Workflow
  1. Executes target application once
  2. Studies IMPI statistics generated
  3. Runs the Intel® MPI Benchmarks to tune collectives used (#ranks, message size, algorithm)


intel®
experience
what's inside™