A FIELD GUIDE TO KNL MEMORY MODES AND OPENMP AFFINITY CONTROL

Jeff Hammond

Exascale CoDesign Team
Data Center Group
Intel Corporation

Legal Disclaimer & Optimization Notice

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

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.

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

Title:

A Field Guide to KNL Memory Modes and OpenMP Affinity Control

Abstract:

This talk will describe different usage models for MCDRAM in Intel Xeon Phi processors (Knights Landing) as well as OpenMP affinity control for Intel Xeon Scalable processors (Skylake). I will use the STREAM benchmark and the Parallel Research Kernels to show how performance (especially memory bandwidth) depends on the different settings. The results demonstrate the trade-offs made between performance and programmability in how high-bandwidth memory is exposed to applications and the importance of handling affinity properly when using threads in a NUMA environment.

Bio:

Jeff Hammond is a Senior System Architect at Intel, where his focus is hardware-software co-design in the context of exascale numerical simulations. He contributes to the development of open standards for parallel programming, including MPI, OpenMP, OpenSHMEM, and ISO C++. Previously, Jeff was a computational scientist at Argonne (ALCF), where he did things related to MPI. His academic background is in computational chemistry.

See https://github.com/jeffhammond/ for details.

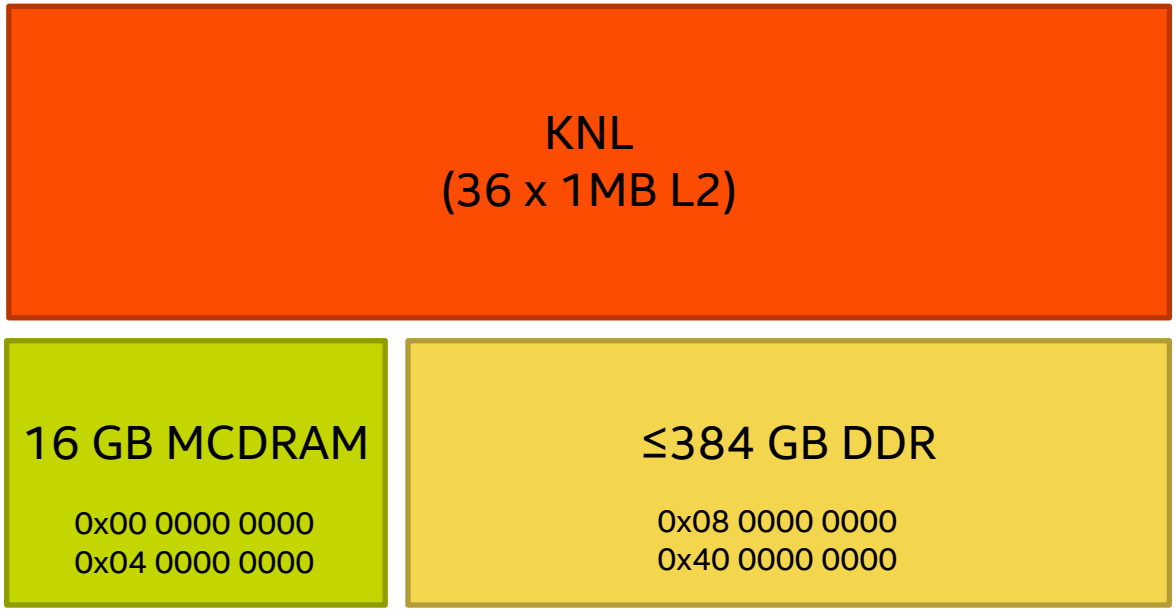
Outline

- Description of KNL memory modes
- STREAM benchmark
  - Cache versus Flat mode
  - Nontemporal stores in KNL
  - Tile versus Core placement in KNL
  - Affinity and Nontemporal stores in SKX
- PRK stencil (2D)
  - DDR vs MCDRAM, affinity in KNL
  - Traditional OpenMP versus Tasking in OpenMP and TBB in KNL
  - Affinity and tasking in SKX

Warning: I am not going to present basic information about CPU hardware (KNL and SKX), Intel C/C++ 2018 or Intel OpenMP affinity environment variables. Please look for documentation online.

MEMORY MODES

Flat Mode



Pro

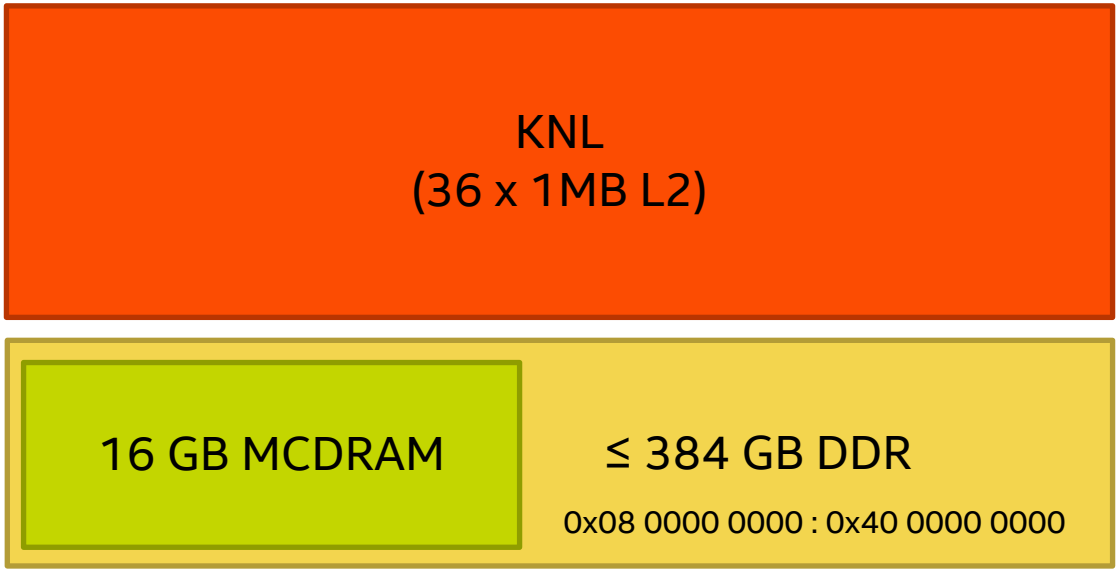
- *Maximum bandwidth and latency performance.*
- Maximum addressable memory.
- No MCDRAM pollution with non-critical data.

Con

- *Software modifications required.*
- Choosing MCDRAM vs DDR hinders application flexibility.

Addresses are given for illustration only. Do not interpret literally.

Cache Mode



Pro

- *No software changes required.*
- Bandwidth benefit (relative to DDR-only).

Con

- *Latency hit to DDR.*
- *Limited sustained bandwidth.*
- All memory is transferred DDR -> MCDRAM -> L2.
- Less addressable memory.

Addresses are given for illustration only. Do not interpret literally.

MCDRAM-only (“lean mode”*)

* This is my name for this talk. Don’t expect anyone else at Intel to understand this term.

Pro

- *Maximum bandwidth and latency performance.*
- *No software changes required.*

Con

- *Memory capacity is limited.*

Addresses are given for illustration only. Do not interpret literally.

NAME

hbwmalloc - The high bandwidth memory interface

SYNOPSIS

```
#include <hbwmalloc.h>
...
int hbw_check_available(void);
void* hbw_malloc(size_t size);
void* hbw_calloc(size_t nmemb, size_t size);
void* hbw_realloc(void *ptr, size_t size);
void hbw_free(void *ptr);
int hbw_posix_memalign(void **memptr, size_t alignment, size_t size);
int hbw_posix_memalign_psize(void **memptr, size_t alignment, size_t size, int pagesize);
int hbw_get_policy(void);
void hbw_set_policy(int mode);
```

> man **https://github.com/memkind/memkind**/blob/dev/man/hbwmalloc.3

I wrote a knock-off for education purposes: https://github.com/jeffhammond/myhbwmalloc

Code Snippets

Heap allocation in C

```
float * fv1 = malloc(sizeof(float) * 1000);
float * fv2 = hbw_malloc(sizeof(float) * 1000);
```

Allocatable arrays in Fortran

```
      REAL, ALLOCATABLE :: A(:), B(:)
!DIR$ ATTRIBUTES FASTMEM :: A
! allocate array 'A' from MCDRAM
     ALLOCATE (A(1:1000))
! allocate array B from DDR
     ALLOCATE  (B(1:1000))
```

Automatic variables (global/common or "stack") will be allocated in DDR in flat mode.

This means you may need to convert from automatic to heap arrays or use hybrid mode if such data is used in a bandwidth-intensive way.

Standard containers in C++ (check docs for details)

```
std::vector<float, hbwmalloc::hbwmalloc_allocator<float> > vec;
```

Choosing the right mode

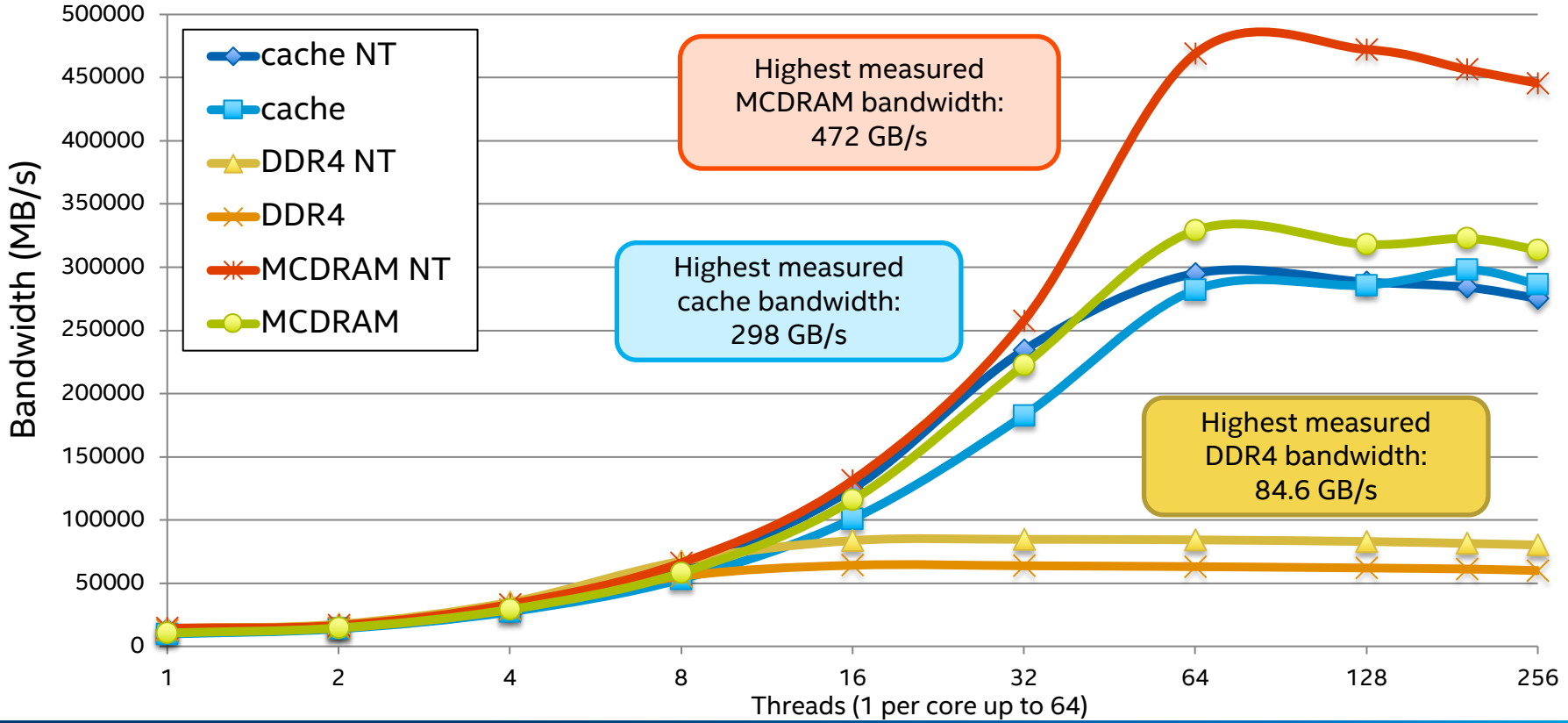
	DDR-only	Lean	Cache	Flat
Software changes	None	None	None	Modify data structures/allocation
Performance	Latency	Bandwidth, Latency	Bandwidth, ~Latency	Bandwidth, Latency
Capacity	DDR	MCDRAM	DDR	DDR+ MCDRAM

~latency: DDR latency include cost of cache miss, unlike flat mode. Cache hit sees MCDRAM latency.

STREAM ON KNL AND SKX

STREAM TRIAD

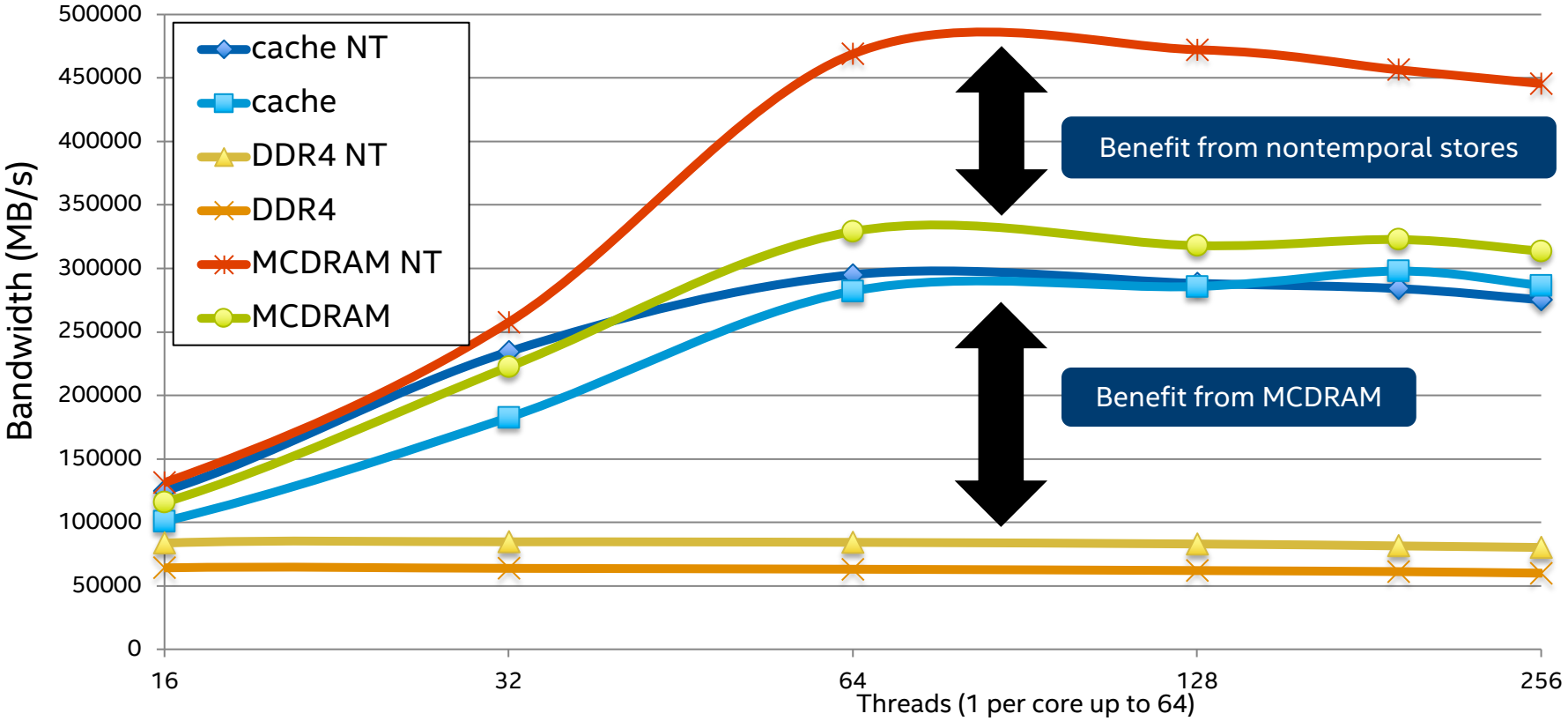
Intel® Xeon Phi™ CPU 7250 @ 1.40GHz
Linux* 3.10.0-327.el7.mpsp_1.3.0.66.x86_64



https://github.com/jeffhammond/STREAM/tree/knl

STREAM TRIAD

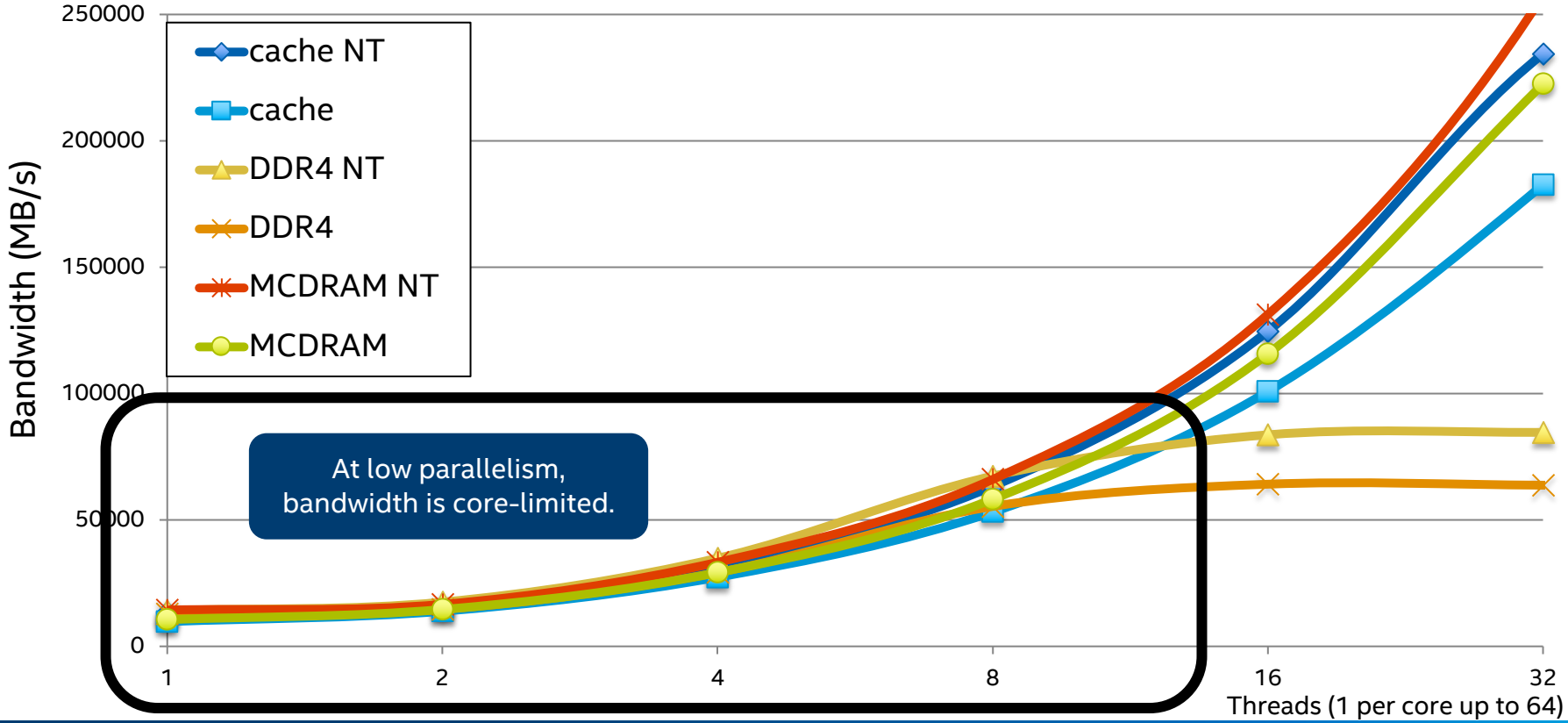
Intel® Xeon Phi™ CPU 7250 @ 1.40GHz
Linux* 3.10.0-327.el7.mpsp_1.3.0.66.x86_64



https://github.com/jeffhammond/STREAM/tree/knl

STREAM TRIAD

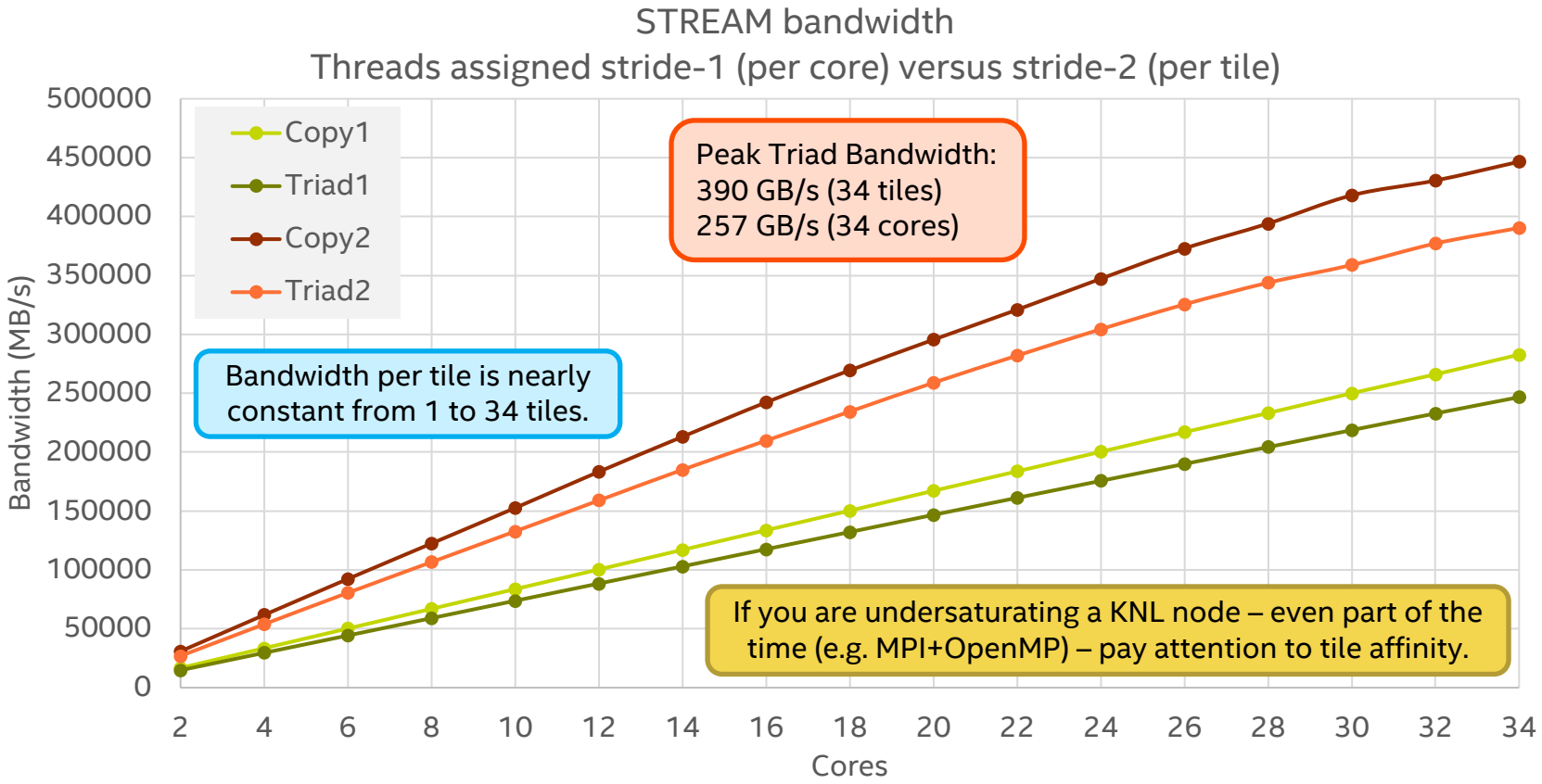
Intel® Xeon Phi™ CPU 7250 @ 1.40GHz
Linux* 3.10.0-327.el7.mpsp_1.3.0.66.x86_64



https://github.com/jeffhammond/STREAM/tree/knl

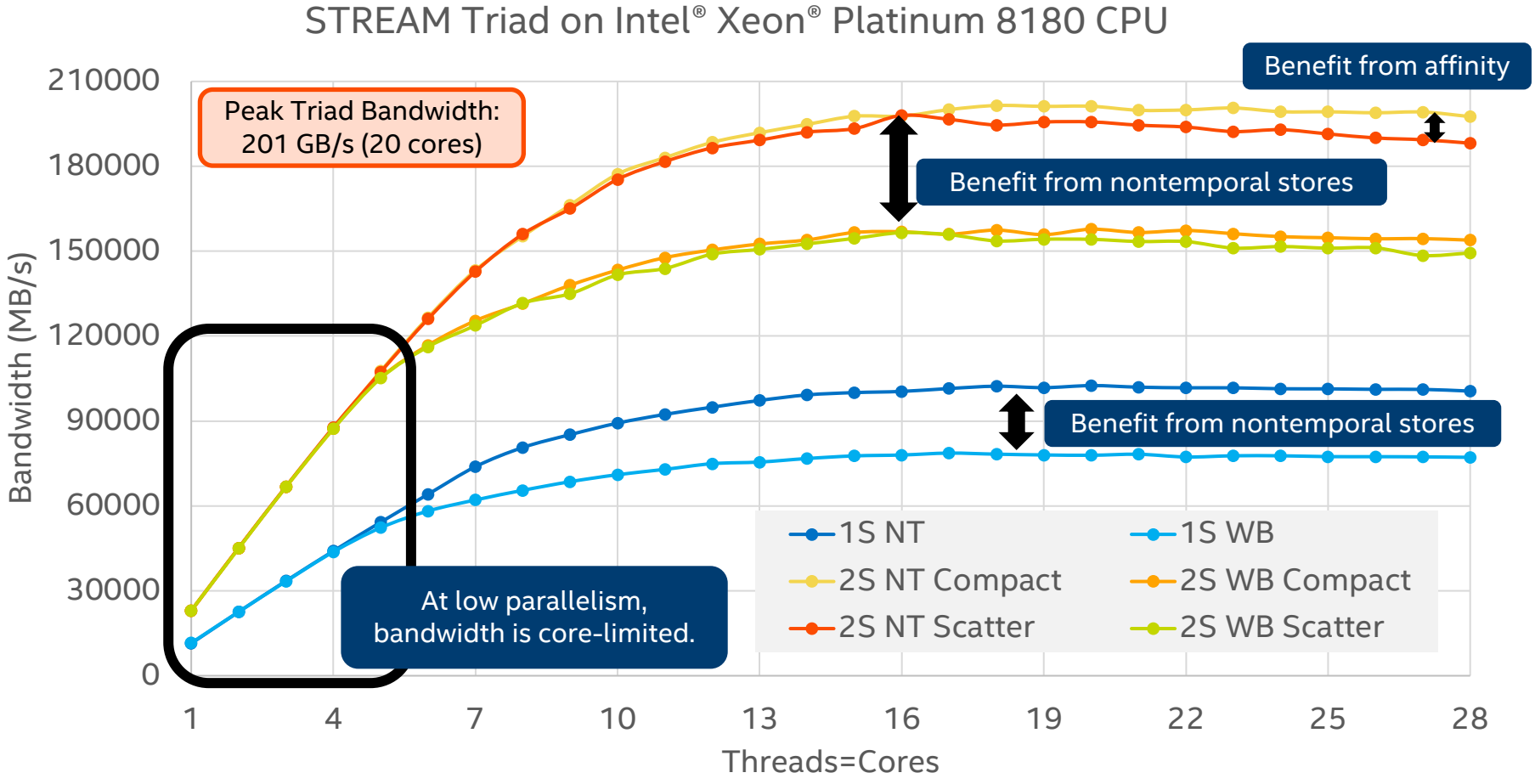
```
# pseudocode for how previous figure data was generated
for n in "quad-flat" "quad-cache" ; do
  <boot node in mode $n>
  for m in 1 0 ; do
    for t in 256 192 128 64 32 16 8 4 2 1 ; do
      for b in stream_c.exe stream_c.nontemporal stream_c.temporal ; do
        export OMP_NUM_THREADS=${t}
        if [ $t -gt 192 ] ; then
          h=4
        elif [ $t -gt 128 ] ; then
          h=3
        elif [ $t -gt 64 ] ; then
          h=2
        else
          h=1
        fi
        export KMP_PLACE_THREADS="1s$((${t}/${h}))c${h}t" # deprecated
        export KMP_AFFINITY=compact,granularity=fine
        numactl -m ${m} ./${b} >& ${b}.${KMP_PLACE_THREADS}.m${m}.${n}.log
      done
    done
  done
done
```

```
# KNL build options
CC = icc
CFLAGS = -O3 -xMIC-AVX512 -qopenmp \
         -mcmodel=medium -shared-intel \
         -DSTREAM_ARRAY_SIZE=134217728 \
         -DOFFSET=0 -DNTIMES=100

TEMPORAL = -qopt-streaming-stores never
NONTEMPORAL = -qopt-streaming-stores always

all: stream_c.temporal stream_c.nontemporal

stream_c.temporal: stream.c
    $(CC) $(CFLAGS) $(TEMPORAL) stream.c -o stream_c.temporal

stream_c.nontemporal: stream.c
    $(CC) $(CFLAGS) $(NONTEMPORAL) stream.c -o stream_c.nontemporal
```



Intel® Xeon Phi™ CPU 7250 @ 1.40GHz, Linux* 3.10.0-327.el7.mpsp_1.3.1.45.x86_64

```
# how to generate data from previous slide
for c in {34..2..2} ; do
  for s in 1 2 ; do
    OMP_PROC_BIND=TRUE \
    OMP_NUM_THREADS=${c} \
    OMP_PLACES="{0}:${c}:${s}" \
    numactl -m 1 ./stream_c.exe
  done
done
```



NT = nontemporal aka streaming stores WB = write back aka traditional stores
1S = 1 socket 2S = 2 socket

```
# runs a superset of what is shown on the previous figure
for a in compact scatter ; do
  for store in temporal nontemporal ; do
    for s in 1 2 ; do
      for c in {28..1..1} ; do
        for t in 1 2 ; do
          KMP_AFFINITY=granularity=fine,${a} \
          KMP_HW_SUBSET=1${s},${c}c,${t}t \
          ./stream_c.${store}
      done
    done
  done
done
```

```
# SKX build options
CC = icc
CFLAGS = -O3 -xCORE-AVX512 -qopt-zmm-usage=low -qopenmp \
         -mcmodel=medium -shared-intel \
         -DSTREAM_ARRAY_SIZE=134217728 \
         -DOFFSET=0 -DNTIMES=100

TEMPORAL = -qopt-streaming-stores never
NONTEMPORAL = -qopt-streaming-stores always

all: stream_c.temporal stream_c.nontemporal

stream_c.temporal: stream.c
    $(CC) $(CFLAGS) $(TEMPORAL) stream.c -o stream_c.temporal

stream_c.nontemporal: stream.c
    $(CC) $(CFLAGS) $(NONTEMPORAL) stream.c -o stream_c.nontemporal
```

Summary

- KNL
  - Performance: MCDRAM (flat) > MCDRAM (cache) >> DRAM
  - Programmability: MCDRAM (cache) = DRAM = MCDRAM (flat) up to 16GB
  - Programmability: MCDRAM (cache) = DRAM > MCDRAM (flat) beyond 16 GB
- Both
  - Nontemporal stores are important for streaming bandwidth workloads.
  - Parallelism is required to saturate bandwidth

Programming nontemporal stores

```
#pragma vector nontemporal
for (i=0; i<length; i++) {
  B[i] = A[i];
}
```

Intel, Cray and PGI compilers support a directive to generate nontemporal stores.

```
#pragma omp simd nontemporal
for (i=0; i<length; i++) {
 B[i] = A[i];
}
```

OpenMP 5 adds nontemporal clause to simd.

```
__m512i t;
for (i=0; i<n; i+=8) {
  t = _mm512_stream_load_si512(&(a[i]));
  _mm512_stream_si512(&(b[i]), t);
}
_mm_sfence();
```

Intel intrinsics (ICC, GCC, Clang) support streaming load/store for every ISA variant. You must close epoch with SFENCE!!!

PARALLEL RESEARCH KERNELS

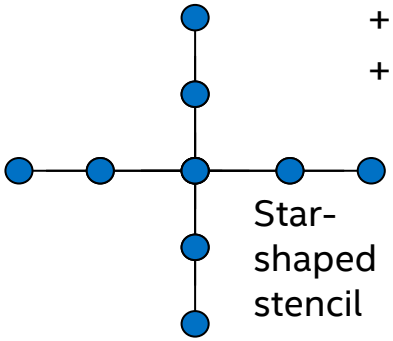
See https://github.com/ParRes/Kernels/tree/master/Cxx11 for details.

Overview of the PRK project

- Reference implementations of common HPC patterns (“micro-apps”)
  - Stencil, Transpose, Nstream (triad), Wavefront, … PIC.
- Lots of different languages and programming models:
  - Octave, Python, Julia, Chapel, Rust, C89, C11, Fortran 2008, C++17
  - MPI1, MPI3, SHMEM, UPC, OpenMP 4.5, TBB, OpenCL, SYCL, …
- Flexible parameterization with solution verification.
- Not tuned for any architecture or problem size!
- Use programming models idiomatically (as best we know how).

Stencil

```
B[2:n-2,2:n-2] += W[2,2] * A[2:n-2,2:n-2]
                + W[2,0] * A[2:n-2,0:n-4]
                + W[2,1] * A[2:n-2,1:n-3]
                + W[2,3] * A[2:n-2,3:n-1]
                + W[2,4] * A[2:n-2,4:n-0]
                + W[0,2] * A[0:n-4,2:n-2]
                + W[1,2] * A[1:n-3,2:n-2]
                + W[3,2] * A[3:n-1,2:n-2]
                + W[4,2] * A[4:n-0,2:n-2]
```



Star-shaped stencil

- Proxy for structured mesh codes. 2D stencil to emphasize non-compute.
- Supports arbitrary radius star and square grid stencils via code generator with constant coefficients.

The experiments here use star with radius=4.

OpenMP implementation

```
void star2(const int n, const int t, std::vector<double> & in, std::vector<double> & out) {
  #pragma omp for collapse(2)
  #pragma omp taskloop collapse(2) firstprivate(n) shared(in,out) grainsize(1)
  for (auto it=2; it<n-2; it+=t)
    for (auto jt=2; jt<n-2; jt+=t)
      for (auto i=it; i<std::min(n-2,it+t); ++i) {
        #pragma omp simd
        for (auto j=jt; j<std::min(n-2,jt+t); ++j) {
          out[i*n+j] += +in[(i+-2)*n+(j+0)] * -0.125 + in[(i+-1)*n+(j+0)] * -0.25
                        +in[(i+0)*n+(j+-2)] * -0.125 + in[(i+0)*n+(j+-1)] * -0.25
                        +in[(i+0)*n+(j+1)] * 0.25 + in[(i+0)*n+(j+2)] * 0.125
                        +in[(i+1)*n+(j+0)] * 0.25 +in[(i+2)*n+(j+0)] * 0.125;
        }
      }
}
```

TBB implementation

```
void star2(const int n, const int t, std::vector<double> & in, std::vector<double> & out) {
  tbb::blocked_range2d<int> range(2, n-2, t, 2, n-2, t);
  tbb::parallel_for( range, [&](decltype(range)& r ) {
    for (auto i=r.rows().begin(); i!=r.rows().end(); ++i ) {
      #pragma omp simd
      for (auto j=r.cols().begin(); j!=r.cols().end(); ++j ) {
          out[i*n+j] += +in[(i+-2)*n+(j+0)] * -0.125 + in[(i+-1)*n+(j+0)] * -0.25
                        +in[(i+0)*n+(j+-2)] * -0.125 + in[(i+0)*n+(j+-1)] * -0.25
                        +in[(i+0)*n+(j+1)] * 0.25 + in[(i+0)*n+(j+2)] * 0.125
                        +in[(i+1)*n+(j+0)] * 0.25 + in[(i+2)*n+(j+0)] * 0.125;
      }
    }
  }, tbb_partitioner );
}
```

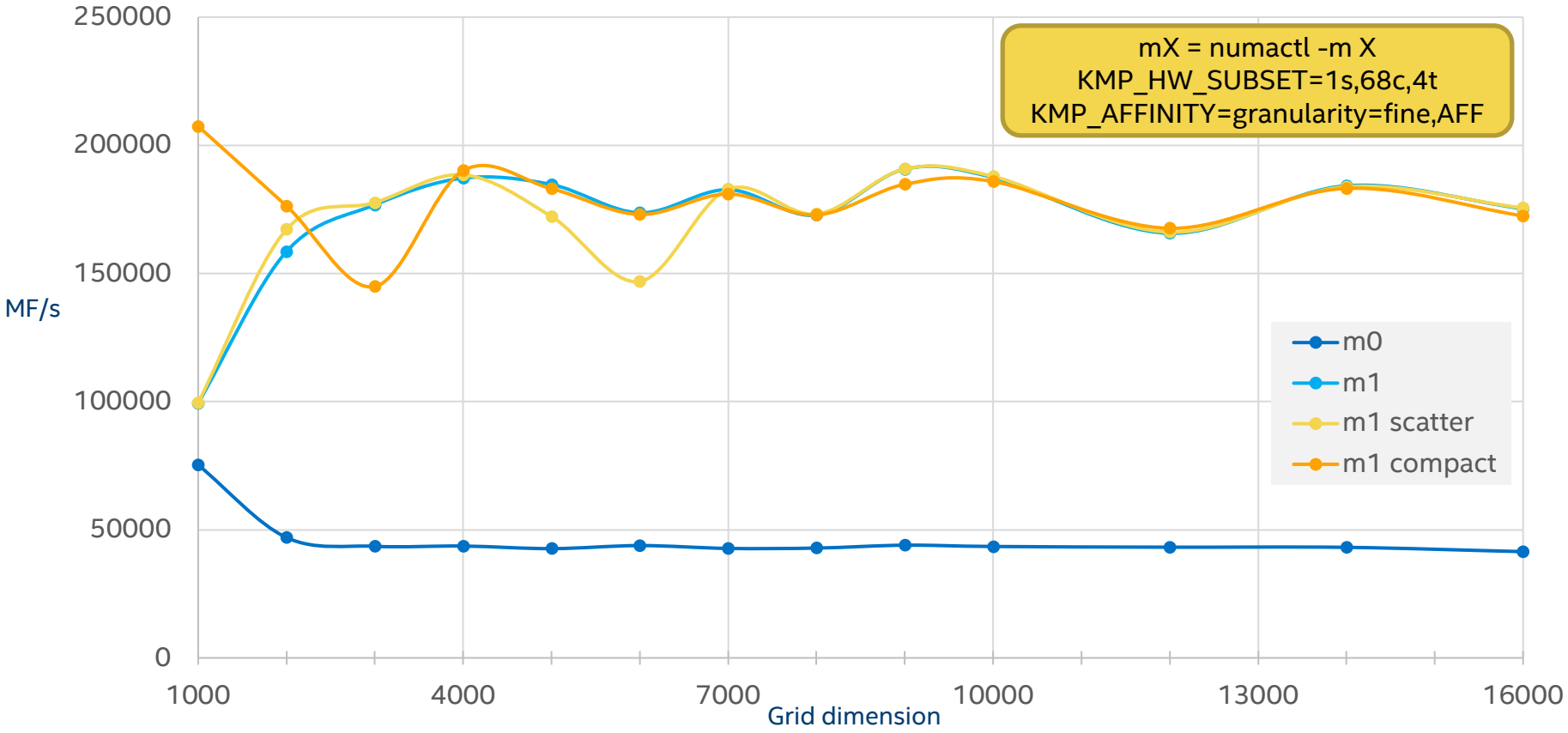
KNL Details

- PRK stencil with pattern=star and radius=4 (default)
- Processor: Intel® Xeon Phi™ CPU 7250 @ 1.40GHz
- Memory: quadrant flat mode.
- Compiler: Intel C/C++ 18.0.1 20171018 (version 18 update 1)
- Settings: 1000 iterations of each after a dummy job to warm up the node
- Blocksize of 64, grainsize of 1 or 10.

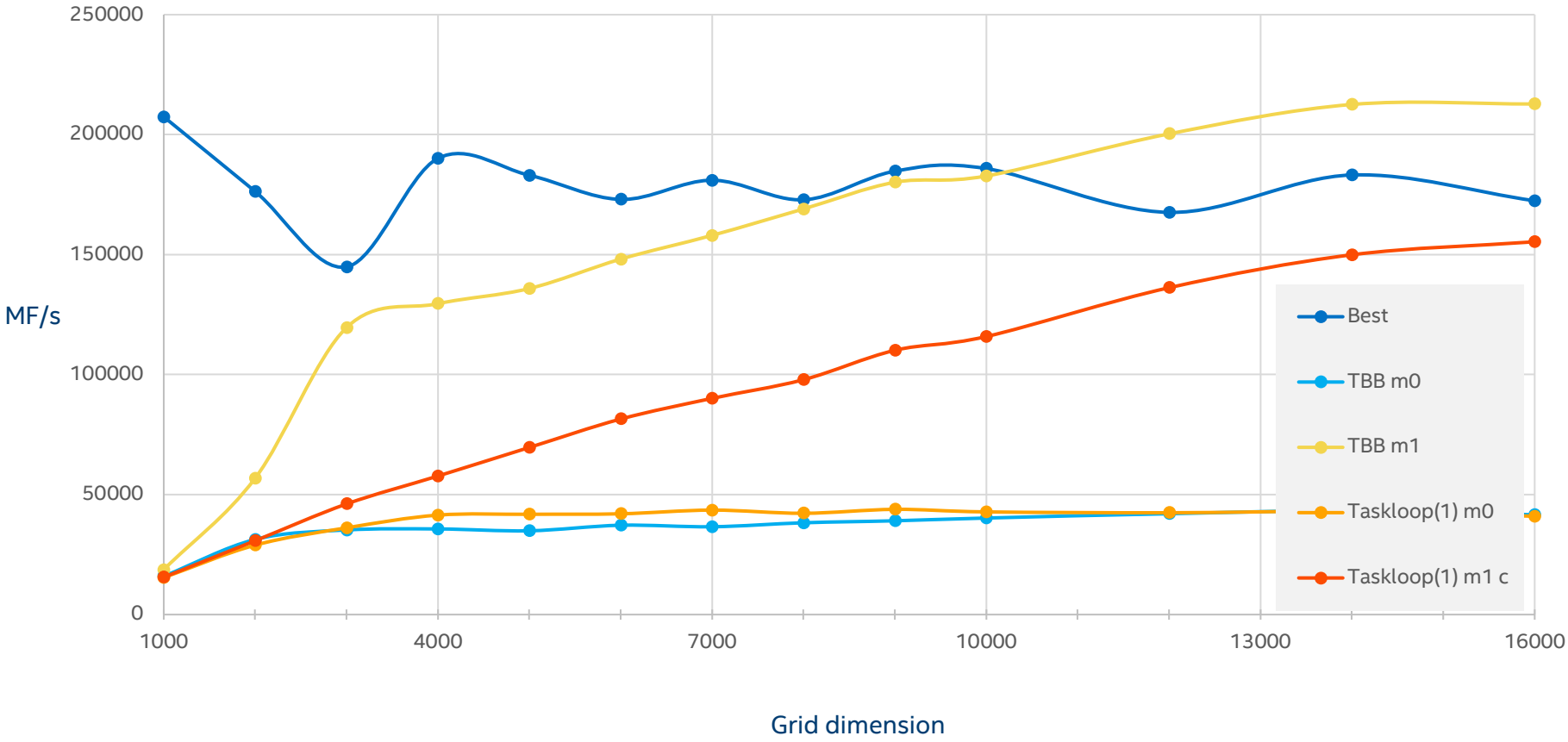
SKX Details

- PRK stencil with pattern=star and radius=4 (default)
- Processor: Intel® Xeon™ Platinum 8180 CPU @ 2.50GHz
- Compiler: Intel C/C++ 18.0.1 20171018 (version 18 update 1)
- Settings: 300 iterations of each after a dummy job to warm up the node
- Blocksize of 32, grainsize of 1 or 10.
- Compiler target: CORE-AVX512 with zmm=low.

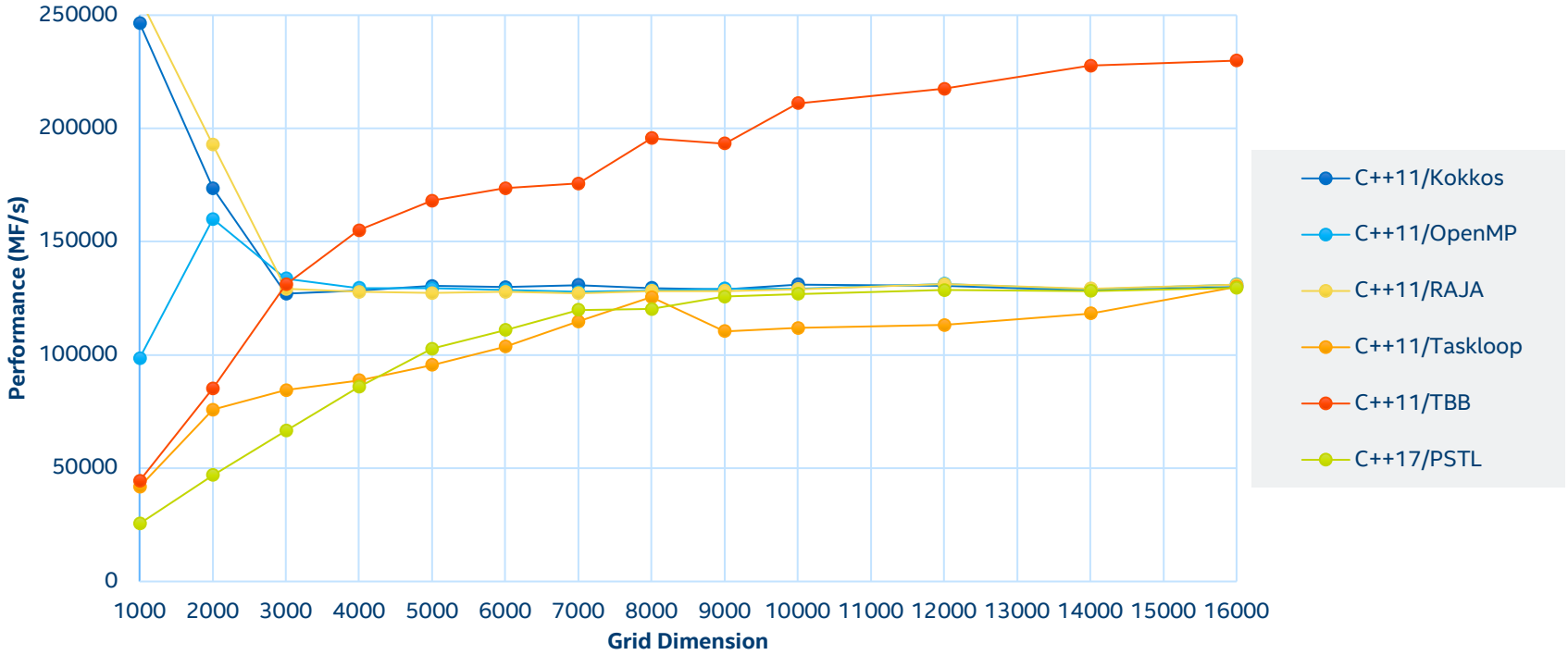
PRK stencil - OpenMP - Intel Xeon Phi 7250



PRK stencil - OpenMP - Intel Xeon Phi 7250

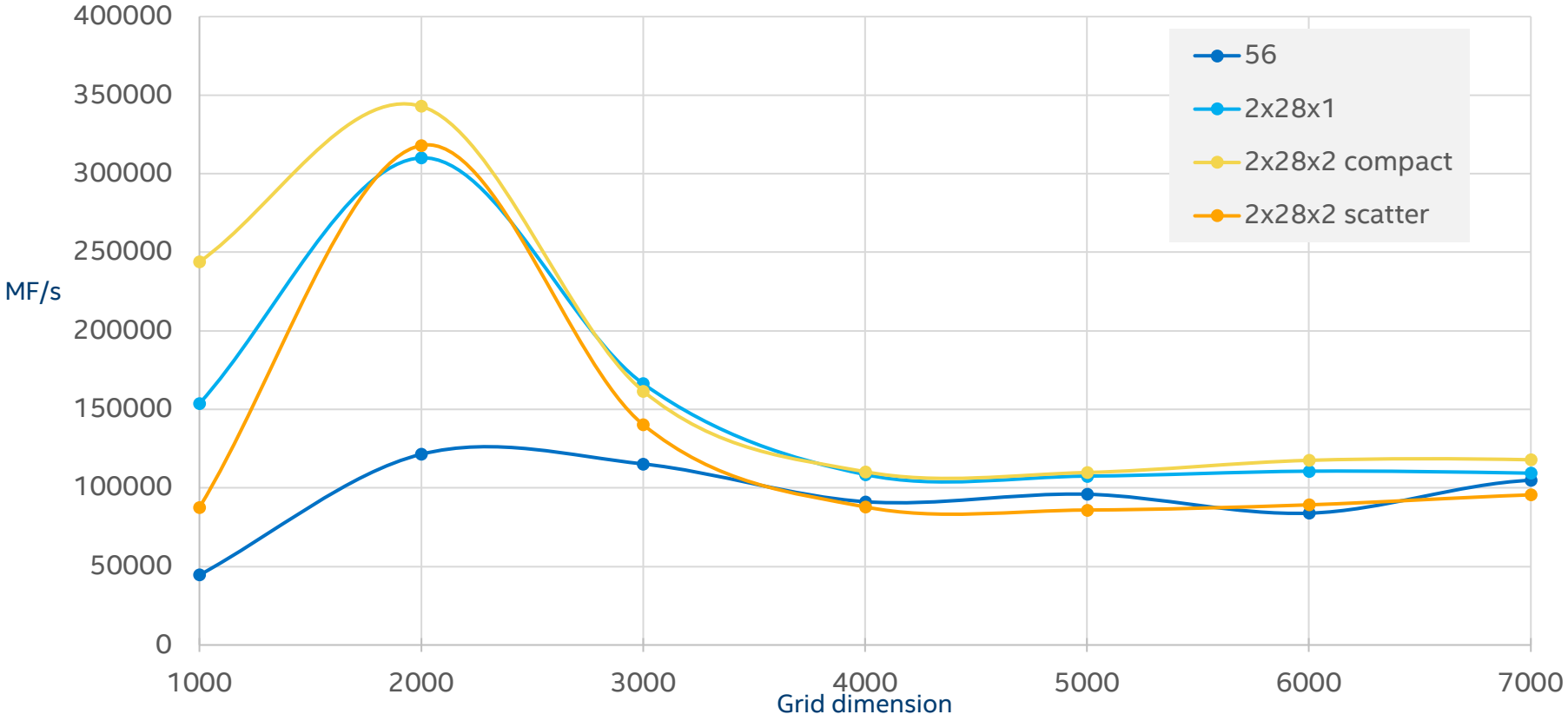


PRK stencil on KNL *before blocking*

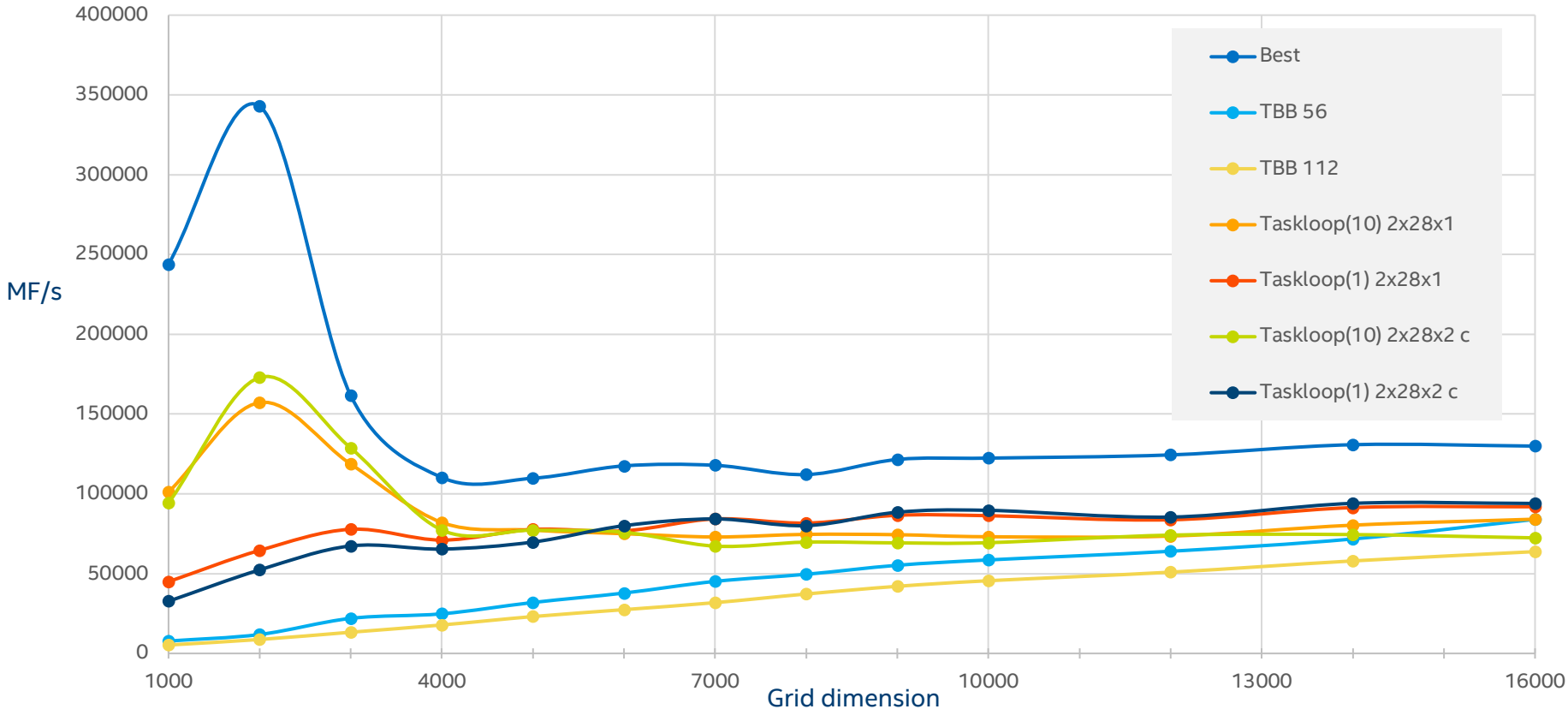


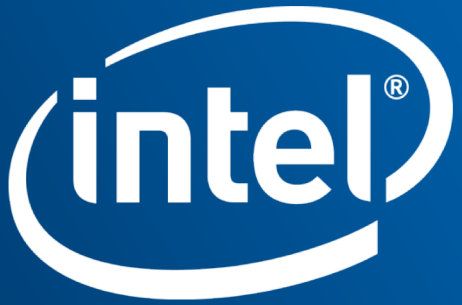
Cilk results are anomalous in C++ due to compiler issue. The C results are good.

PRK stencil - OpenMP - Intel Xeon Platinum 8180



PRK stencil - OpenMP - Intel Xeon Platinum 8180




intel®