
INTEL® HPC DEVELOPER CONFERENCE
FUEL YOUR INSIGHT
intel

INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

UNDERSTANDING AND HARNESSING THE CAPABILITIES OF INTEL® XEON PHI™ PROCESSOR (CODE NAMED KNIGHTS LANDING)

Sumedh Naik, James Tullos, Antonio Valles, Michael Hebenstreit,
Mallick Arigapudi, Zakhar Matveev

Intel Corporation

November 2016

Agenda

Logistics

Overview of Intel® Xeon Phi™ Processor

Parallelism

Vectorization

Memory Bandwidth

Summary

Additional Resources

Logistics

Multiple presenters, each with different expertise

If you have questions not directly related to the topic being presented, please get the attention of one of the non-presenting coordinators

If you need assistance setting up software, we can do that separately from the presenter

We have some USB sticks with PuTTY, WinSCP, and VNC Viewer installers if needed

Use Cottonwood_Lab wifi (utah1614)

Endeavour Access Instructions

SSH to 207.108.8.123

Use login info provided on strip

Use screen

Nodes are paired.

- Every pair has one node in MCDRAM Flat and one in MCDRAM Cache. Coordinate with the person next to you.
- The first two labs run in either mode, MCDRAM mode you will alternate.

Use /nfs/scratch2/KNL_Lab_students

- Create subdirectory for yourself

Run *vncserver* while on login node (ecompile5), get port from log file

Open SSH tunnel between laptop and hostname:VNC_port

- In PuTTY, this is under Change Settings->Connection->SSH->Tunnels
  - Remember to click Add!
  - Recommend high port number (add 50000 to remote port number) to avoid conflicts

Run VNC Viewer and connect to the local port specified previously

OVERVIEW OF INTEL® XEON PHI™ PROCESSOR

The 2nd Generation Intel® Xeon Phi™ Processor (code named Knights Landing)

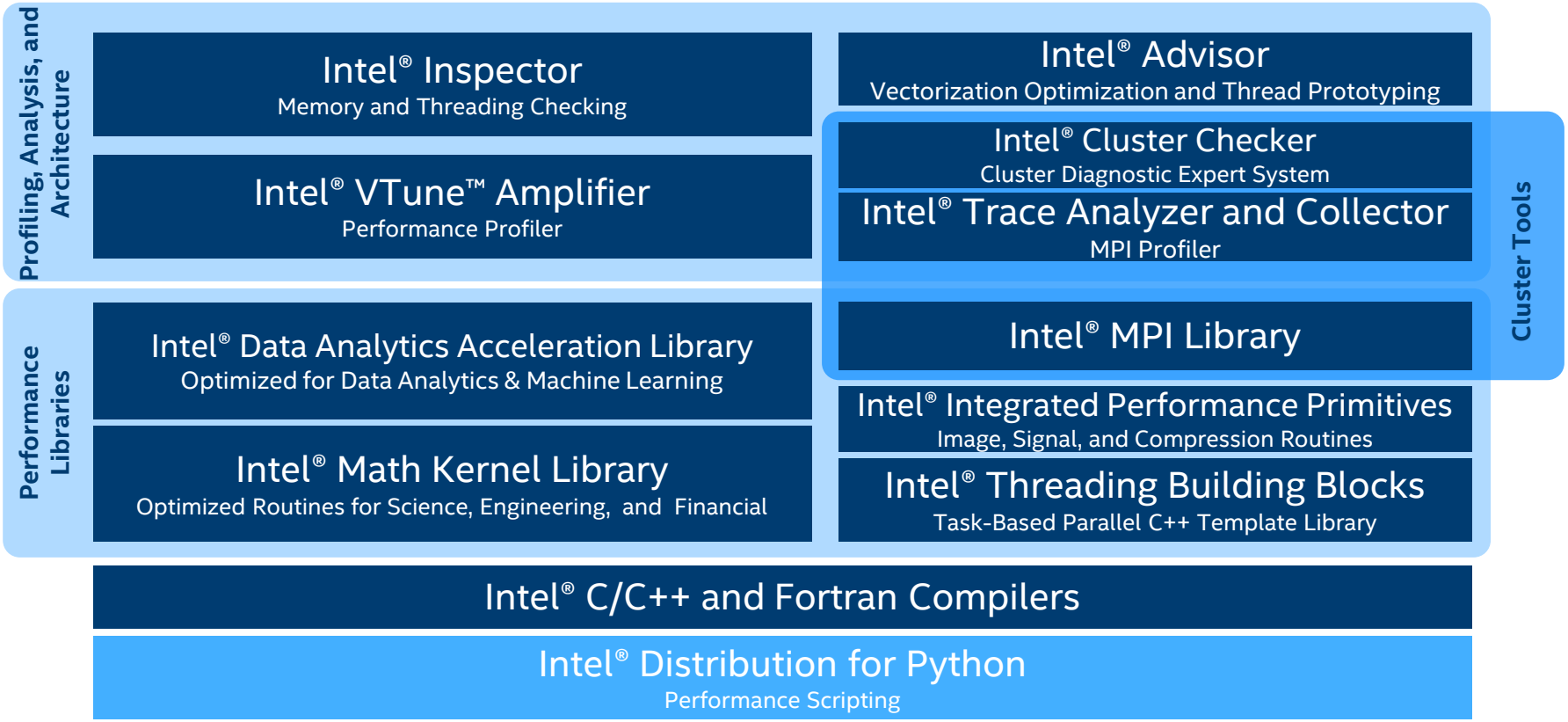
Targeted for high performance computing

- **High BW**
  - Integrated memory on package: 490 measured GB/sec*; up to 16 GB capacity
  - Cache or separate NUMA node
- **Cluster Parallelism**
  - Integrated fabric on package (Omni-Path)
  - 2x100 Gbps ports
- **Thread level Parallelism (TLP)**
  - Up to 68 cores X 4 hyper-threads per core = 272 threads (7290 offers 72 cores; premium part)
  - Tiles: 2 cores per tile sharing Cache-Home-Agent for Cache Coherency and 1MB MB L2 cache
- **Data-level Parallelism (DLP)**
  - Introduces AVX-512 ISA
  - Compatible with previous ISA (AVX, SSE, …)
- **Instruction-level Parallelism (ILP)**
  - Out-of-order core
  - Two vector processing units per core
- **Power Efficiency**
  - 215 Watts TDP (7290 is 245 Watts)
  - 2x145 Watts TDP for Xeon Dual socket BDW E5-2697 (2x18 cores)

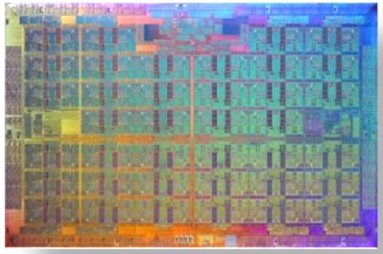
Performance:
Vector Peak Performance: 3+TF DP, 6+TF SP
Bandwidth: 490 GB/sec Triad Stream Score*

*Using Streaming Stores in Flat Mode

Intel® Parallel Studio XE



Harnessing the capabilities of Intel Xeon Phi



For best experience on Intel Xeon Phi we especially need to focus on three items

- Parallelism
- Vectorization
- Memory BW

This lab, will cover these items in detail

PARALLELISM

Parallelism Lab Learning Goals

Benefits of MPI vs. Threading

Understanding of imbalance and how it is measured

Usage of Intel® Software Development Tools to find performance bottlenecks

MPI vs. Threading

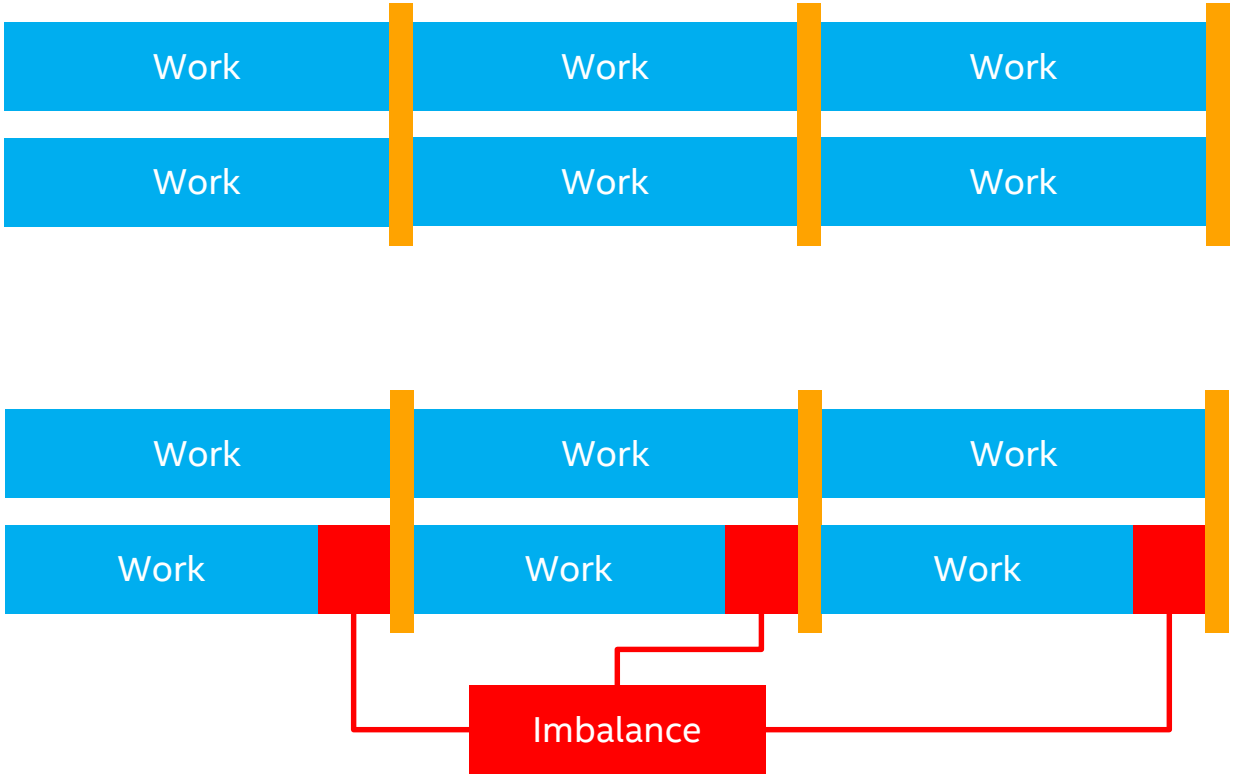
MPI

- Distributed memory – individual processes have no visibility to data in other processes
  - Can grant visibility using MPI RMA
- Increased memory usage
- Explicit data flow control
- Scales across multiple systems

Threading

- Shared memory – individual threads can have visibility to same data space
  - Or can use private data spaces
- Data flow is implicit, and can be hidden
- Scales across multiple cores, but within single memory space

Defining Imbalance in Parallelism



Lab Code/Overview

Cardiac Demo (https://github.com/CardiacDemo/Cardiac_demo)

- Electro-cardiograph simulation code
- Hybrid MPI+OpenMP

Run code standalone first to get baseline

Run under various tools to find performance bottlenecks

Run with *numactl –p 1*

- On MCDRAM Flat, will take advantage of MCDRAM, not needed (but won't hurt) on MCDRAM Cache.

Lab Steps

Run scripts are provided in *run* subdirectory

Values to adjust in scripts:

- -n <nranks> - Number of MPI ranks
- -genv OMP_NUM_THREADS <nthreads> - Number of OpenMP threads
- -t <time> - Solution time

run_base.sh – basic run, no tools

run_mps.sh – run under MPI Performance Snapshot to collect MPI/OpenMP imbalance data

run_vtune.sh – run under Intel® VTune™ Amplifier XE to investigate non-MPI performance aspects

run_trace.sh – run under Intel® Trace Analyzer and Collector to investigate MPI performance

run_roofline.sh – run under Intel® Advisor to collect roofline analysis data

Lab Discoveries

Running as default in run scripts leads to poor scaling

Analysis steps determine that scaling is due to inefficient MPI communication patterns (all ranks using point-to-point with all other ranks)

Rearranging data to improve locality improves communication patterns (all ranks only communicate with neighboring ranks)

- Add *–i* to heart_demo options

VECTORIZATION

Vectorization Lab Learning Goals

Understand how to enable Xeon Phi AVX-512 vectorization

Usage of Intel® Software Development Tools to analyze and improve vectorization

- Compiler optimization report
- Annotated Assembly
- Introduction to Vector Advisor

Understand how to work around common obstacles to vectorization

Lab Code/Overview

Two labs

- Beginner lab in C++
  - Given an array, find the minimum and maximum element and their index
- Advanced lab in Fortran
  - Example loop from CAM-SE CORAL RFP

Run code standalone first to get baseline

- Modify source code and use Intel® software development tools to enable, analyze, and improve vectorization

First Lab

Goal: Vectorize loop

- Use Intel compiler:
  - Source /nfs/scratch2/KNL_LAB/endeavour_prep.sh
- Compiler switches
  - –xmic-avx512 for Xeon Phi AVX512
  - –qopt-report=5 for opt report
  - –S and –g for annotated ASM listing
  - -O3 for advanced optimizations
  - -std=c++11 for random numbers
  - -debug inline-debug-info for improved debug info

```
30 void testVer ( float *arr, int arr_len, float& elem, int& Idx, bool bMin )
31 {
32    elem = arr[0];
33    Idx = 0;
34    for ( int i = 1; i < arr_len; i++ )
35    {
36
37       if ( bMin == true )
38       {
39          if (elem > arr[i] )
40          {
41             elem = arr[i];
42             Idx = i;
43          }
44       } else {
45          if (elem < arr[i] )
46          {
47             elem = arr[i];
48             Idx = i;
49          }
50       }
51    }
52 }
```

Lab1.cpp
minMax_1.cpp
./build.sh

Making changes

cp minMax_1.cpp to minMax_<N>.cpp (change testVer)
modify build.sh: testVer=<N>
execute ./build.sh to generate
- minMax_<N>.asm
- minMax_<N>.optrpt
- lab1_<N> executable

Vectorization Tip for Lab 1

By default won't vectorize if compiler may generate incorrect code

- Anti-Dependence: Write-After-Read
  - For { a[i] = b[i]; b[i] = 3; }
  - For { a[i-1] = a[i] + 3; }
- Flow-Dependence: Read-After-Write
  - For {b[i] = 3; a[i] = b[i]; }
  - For { a[i] = a[i-1] + 3; }
- Workarounds
  - User has to inspect code and verify vectorization can be done
    - Before loop: #pragma ivdep; #pragma vector always; #pragma omp simd

Compiler found assumed dependence

Pragmas discard assumed dependence. If assumed dependence is real and user used pragmas, results will be wrong

Hints:

Ivdep – discard assumed dependence
Vector always – vector even if inefficient

Command:

Omp simd – vectorize (no dependence analysis
Nor efficiency analysis to restrict vectorization)

```
LOOP BEGIN at minMax_1.cpp(34,4)
<Predicate Optimized v1>
   remark #25422: Invariant Condition at line 37 hoisted out of this loop
   remark #15344: loop was not vectorized: vector dependence prevents vectorization
   remark #15346: vector dependence: assumed ANTI dependence between arr[i] (41:13)
 and *elem (41:13)
   remark #15346: vector dependence: assumed FLOW dependence between *elem (41:13)
and arr[i] (41:13)
   remark #25439: unrolled with remainder by 2
LOOP END
```

Opt Report
Original Version
lab1_1

Vectorization Tip 2 for Lab 1

Dependency chains from function parameters may inhibit vectorization

- Workarounds
  - Break dependency chains by using local variables
  - Use local variables inside loops
  - Set pointers equal to local variables after loop complete
  - In our case elem and Idx are the pointers / references used inside loop

```
void testVer ( float *arr, int arr_len, float& elem, int& Idx, bool bMin )
{
  //break dependency by using local variables
  float myElem = arr[0];
  int myIdx = 0;
  #pragma omp simd
  for ( ...) {
    ... // use myElem and use myIdx here
   }
   // set pointers to equal local variables
   ...
}
```

Lab1_2

Vectorization Tip 3 for Lab 1

```
LOOP BEGIN at minMax_3.cpp(35,4)
<Predicate Optimized v1>
   remark #25422: Invariant Condition at line 38 hoisted out of this loop
   remark #15316: simd loop was not vectorized: scalar assignment in simd loop
 is prohibited, consider private, lastprivate or reduction clauses    [ minMax_
3.cpp(42,13) ]
   remark #15552: loop was not vectorized with "simd"
   remark #25439: unrolled with remainder by 2
LOOP END
```

Opt Report Lab1_3

Branch is limiting vectorization

- Compiler tells us to try a reduction clause
- Workarounds
  - We want to find the min element or the max element in the loops
  - Use following reduction clauses: reduction(min:myElem); reduction (max,myElem)
  - Problem is we can't use the different clauses on the same line

Solution

Further optimization could be to align array. We will discuss alignment in next lab

Homework:

1. If you wanted to combine SIMD with omp parallel for, why would this be insufficient for this loop?

```
#pragma omp parallel for simd reduction(max:myElem)
```

2. Why does alignment not give large performance increase in this case? <hint: code vectorizes into three loops: peel-loop, main-loop, and remainder-loop. What does peel loop do?>

```
void testVer ( float *arr, int arr_len, float& elem, int& Idx, bool bMin )
{
   float myElem = arr[0];
   int myIdx = 0;
   if ( bMin == true )
   {
      #pragma omp simd reduction(min:myElem)
      for ( int i = 1; i < arr_len; i++ )
      {
         if (myElem > arr[i] )
         {
            myElem = arr[i];
            myIdx = i;
         }
      }
   } else {
      #pragma omp simd reduction(max:myElem)
      for ( int i = 1; i < arr_len; i++ )
      {
         if (myElem < arr[i] )
         {
            myElem = arr[i];
            myIdx = i;
         }
      }
    }
    Idx = myIdx;
    elem = myElem;
}
```

Inspect ASM Listing - I

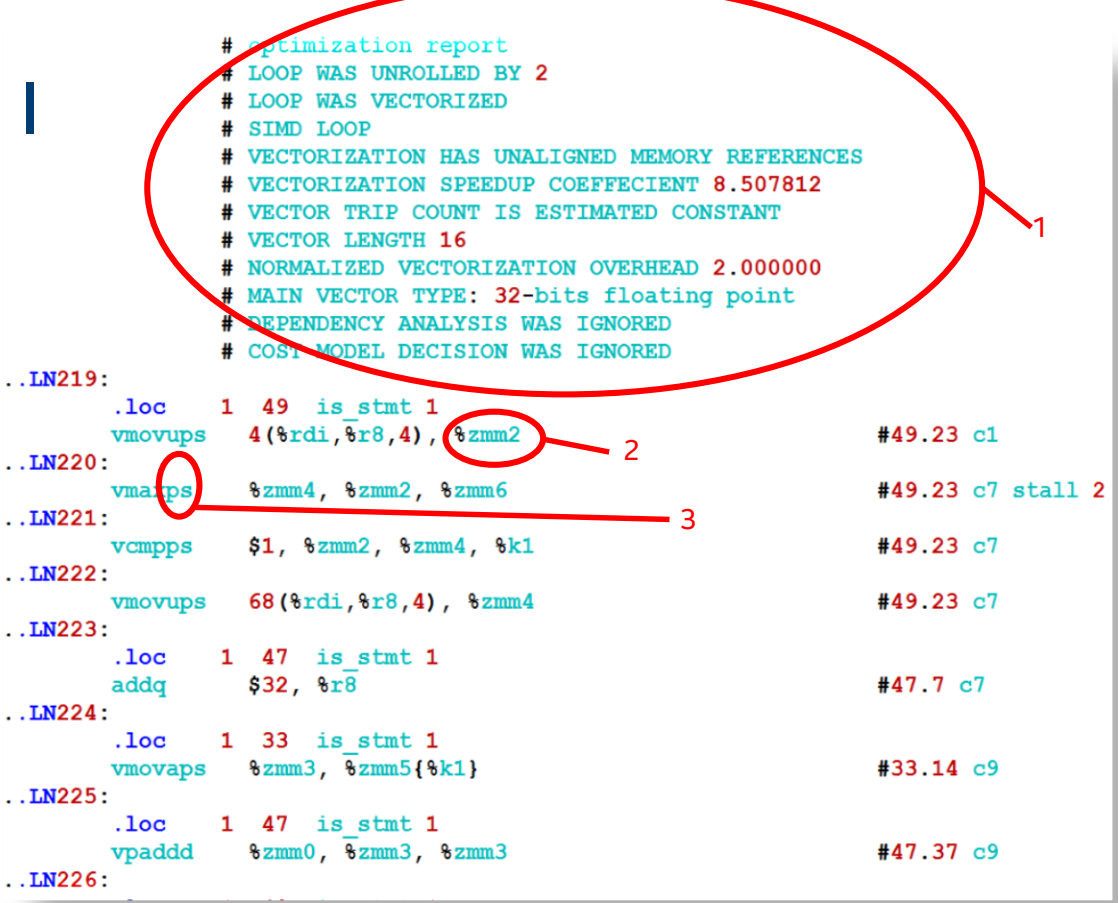
Did code vectorize with AVX-512 ISA?

1: Vectorization Annotation

- Vectorized, unrolled by 2
- Vector length of 2
- *FYI: Unaligned memory reference*

2: Using AVX-512 registers (ZMM)

3: Using Packed-Single instructions 'ps' instead of Scalar-Single




```
# optimization report
# LOOP WAS UNROLLED BY 2
# LOOP WAS VECTORIZED
# SIMD LOOP
# VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
# VECTORIZATION SPEEDUP COEFFECIENT 8.507812
# VECTOR TRIP COUNT IS ESTIMATED CONSTANT
# VECTOR LENGTH 16
# NORMALIZED VECTORIZATION OVERHEAD 2.000000
# MAIN VECTOR TYPE: 32-bits floating point
# DEPENDENCY ANALYSIS WAS IGNORED
# COST MODEL DECISION WAS IGNORED
..LN219:
        .loc    1  49  is_stmt 1
        vmovups   4(%rdi,%r8,4), %zmm2                  #49.23 c1
..LN220:
        vmaxps    %zmm4, %zmm2, %zmm6                   #49.23 c7 stall 2
..LN221:
        vcmpps    $1, %zmm2, %zmm4, %k1                 #49.23 c7
..LN222:
        vmovups   68(%rdi,%r8,4), %zmm4                 #49.23 c7
..LN223:
        .loc    1  47  is_stmt 1
        addq      $32, %r8                              #47.7 c7
..LN224:
        .loc    1  33  is_stmt 1
        vmovaps   %zmm3, %zmm5{%k1}                     #33.14 c9
..LN225:
        .loc    1  47  is_stmt 1
        vpaddd    %zmm0, %zmm3, %zmm3                   #47.37 c9
..LN226:
```

Inspect ASM Listing - II

Compiler provides information per instruction

4: Line number and column number

Before loop:
Current max is really 16 maximums in zmm4
Current max idx is really 16 max indicies in zmm5
Candidate max indicies in zmm3

During loop
- Zmm2 = load of arr[i]
- Zmm6 = max (zmm2,zmm4)
- K-mask vector k1 = Compare (zmm2,zmm4)
- Zmm5 = new max indicies (k1 & zmm3)
- Zmm3 = Increment candidate max indicies by adding zmm3 with zmm0
- …

5: compiler estimates number of cycles operations will take

AVX512 new feature

```
# optimization report
# LOOP WAS UNROLLED BY 2
# LOOP WAS VECTORIZED
# SIMD LOOP
# VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
# VECTORIZATION SPEEDUP COEFFECIENT 8.507812
# VECTOR TRIP COUNT IS ESTIMATED CONSTANT
# VECTOR LENGTH 16
# NORMALIZED VECTORIZATION OVERHEAD 2.000000
# MAIN VECTOR TYPE: 32-bits floating point
# DEPENDENCY ANALYSIS WAS IGNORED
# COST MODEL DECISION WAS IGNORED
..LN219:
        .loc    1  49  is_stmt 1
        vmovups   4(%rdi,%r8,4), %zmm2                  #49.23 c1
..LN220:
        vmaxps    %zmm4, %zmm2, %zmm6                   #49.23 c7 stall 2
..LN221:
        vcmpps    $1, %zmm2, %zmm4, %k1                 #49.23 c7
..LN222:
        vmovups   68(%rdi,%r8,4), %zmm4                 #49.23 c7
..LN223:
        .loc    1  47  is_stmt 1
        addq      $32, %r8                              #47.7 c7
..LN224:
        .loc    1  33  is_stmt 1
        vmovaps   %zmm3, %zmm5{%k1}                     #33.14 c9
..LN225:
        .loc    1  47  is_stmt 1
        vpaddd    %zmm0, %zmm3, %zmm3                   #47.37 c9
..LN226:
```

4

5

Vectorization Lab 2

Goal: Use what you have learned from Lab 1 to vectorize a harder example

- Use Intel compiler & Vector Advisor
- Code is a kernel developed from CAM-SE CORAL RFP: https://asc.llnl.gov/CORAL-benchmarks/

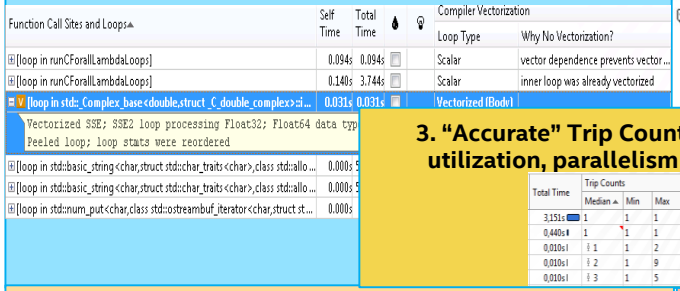
Lab2_1

```
module subroutine test(div)

    real*8, dimension (np,np), intent(inout) :: div

    ! convert to contra variant form and multiply by g
    do j=1,tripCount
      do i=1,tripCount
        gv(i,j,1)=metdet(i,j)*(Dinv(1,1,i,j)*v(i,j,1) + Dinv(1,2,i,j)*v(i,j,2))
        gv(i,j,2)=metdet(i,j)*(Dinv(2,1,i,j)*v(i,j,1) + Dinv(2,2,i,j)*v(i,j,2))
       enddo
    enddo

    ! compute d/dx and d/dy
    do j=1,tripCount
       do k=1,tripCount
          dudx00=0.0d0
          dvdy00=0.0d0
          do i=1,tripCount
             dudx00 = dudx00 + Dvv(i,k  )*gv(i,j  ,1)
             dvdy00 = dvdy00 + Dvv(i,k  )*gv(j  ,i,2)
          end do
          div(k  ,j  ) = dudx00
          vvtemp(j  ,k  ) = dvdy00
       end do
    end do

    do j=1,tripCount
       do i=1,tripCount
          div(i,j)=(div(i,j)+vvtemp(i,j))*(rmetdet(i,j)*rrearth)
       end do
    end do

end subroutine test
```

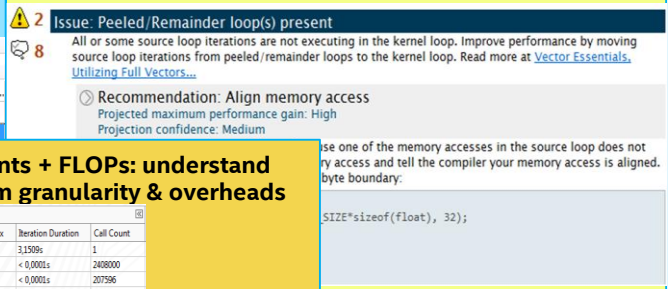
5 Steps to Efficient Vectorization - Vector Advisor

(part of Intel® Advisor, Parallel Studio, Cluster Studio)

1. Compiler diagnostics + Performance Data + SIMD efficiency information

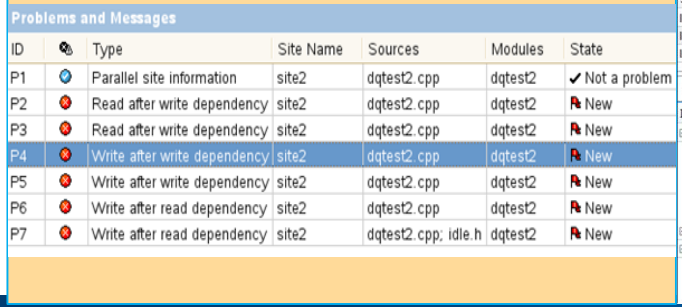


2. Guidance: detect problem and recommend how to fix it

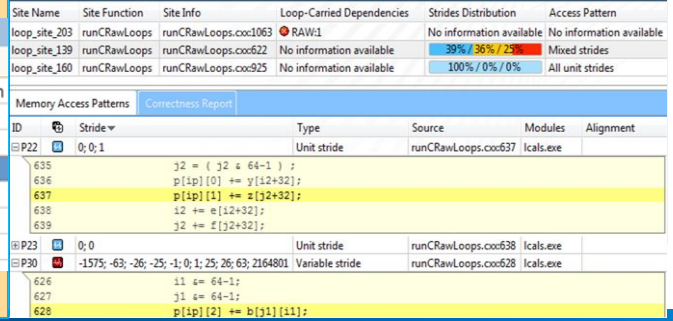


3. "Accurate" Trip Counts + FLOPs: understand utilization, parallelism granularity & overheads

4. Loop-Carried Dependency Analysis

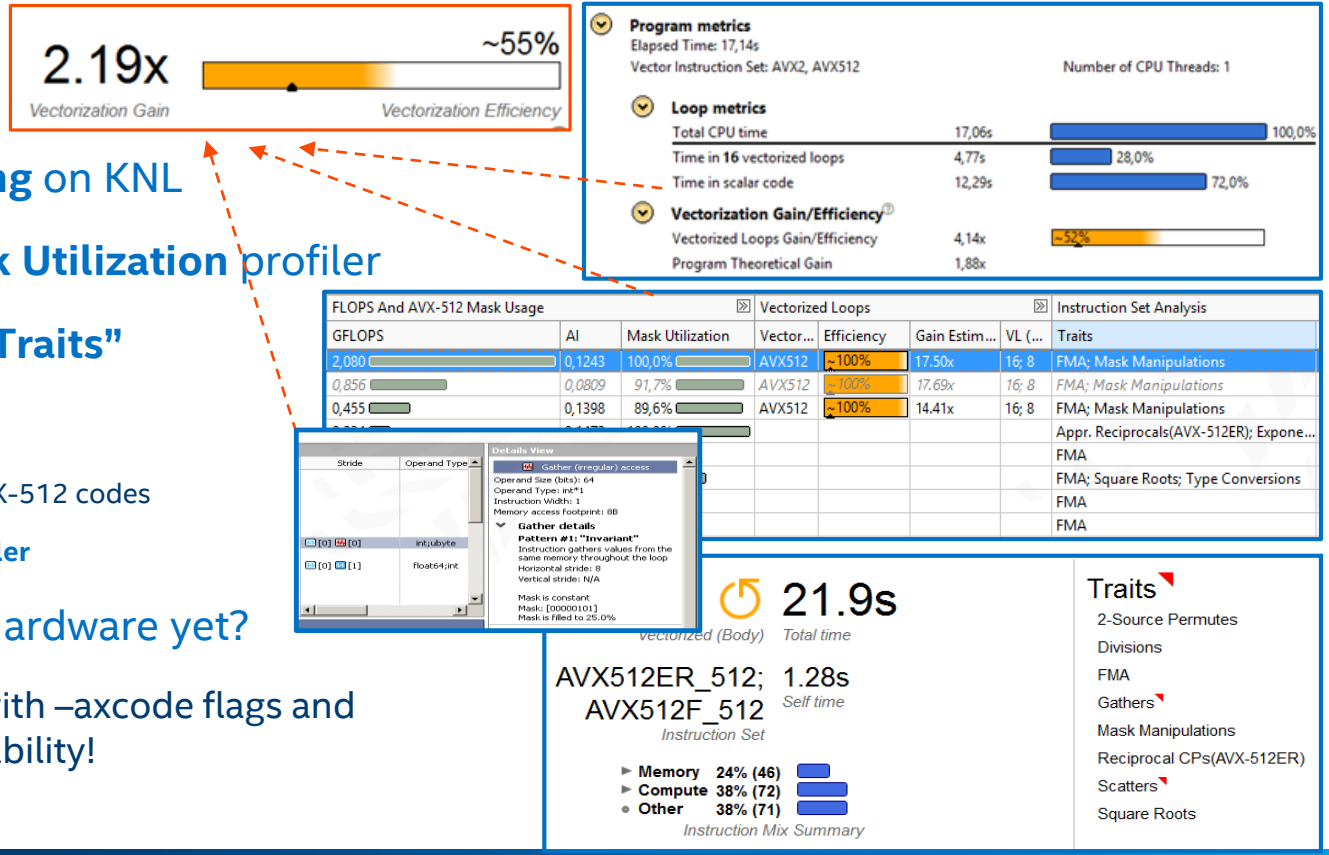


5. Memory Access Patterns Analysis



Advisor 2017: **AVX-512** specific performance insights

- **Native AVX-512 profiling** on KNL
- Precise **FLOPs and Mask Utilization** profiler
- AVX-512 **Advices and "Traits"**
- And more..
  - Performance **Summary for** AVX-512 codes
  - **AVX-512 Gather/Scatter Profiler**
- No access to AVX-512 Hardware yet?
  - Explore AVX-512 code with –axcode flags and new Advisor Survey capability!



Analyze Baseline Version

Focus on first loop for now

- Unaligned memory references
- Access patterns generating gathers:
  - Dinv( ) not accessed with stride of 1

```
# optimization report
# LOOP WAS VECTORIZED
# VECTORIZATION HAS UNALIGNED MEMORY REFERENCES
# VECTORIZATION SPEEDUP COEFFECIENT 2.464844
# VECTOR TRIP COUNT IS ESTIMATED CONSTANT
# VECTOR LENGTH 16
# NORMALIZED VECTORIZATION OVERHEAD 0.140625
# MAIN VECTOR TYPE: 64-bits floating point
# LOOP WITH IRREGULAR ACCESS
```

Memory Access Patterns Report | Dependencies Report | Recommendations

ID	Stride	Type	Source	Nested Function	Variable references
P3		Gather stride	test_kernel_1.F90:20		

```
18      do j=1,tripCount
19        do i=1,tripCount
20          gv(i,j,1)=metdet(i,j)*(Dinv(1,1,i,j)*v(i,j,1) + Dinv(1,2,i,j)*v(i,j,2))
21          gv(i,j,2)=metdet(i,j)*(Dinv(2,1,i,j)*v(i,j,1) + Dinv(2,2,i,j)*v(i,j,2))
22        enddo
```

P4		Gather stride	test_kernel_1.F90:21		

```
19        do i=1,tripCount
20          gv(i,j,1)=metdet(i,j)*(Dinv(1,1,i,j)*v(i,j,1) + Dinv(1,2,i,j)*v(i,j,2))
21          gv(i,j,2)=metdet(i,j)*(Dinv(2,1,i,j)*v(i,j,1) + Dinv(2,2,i,j)*v(i,j,2))
22        enddo
23      enddo
```

Vector Advisor Memory Access Patterns Analysis

Vectorizing Loop 1

Unaligned Accesses

-- how to fix:

```
!dir$ attributes align:64 :: gv
!dir$ attributes align:64 :: v
!dir$ attributes align:64 :: Dinv
!dir$ attributes align:64 :: DinvC
!dir$ attributes align:64 :: metdet
!dir$ attributes align:64 :: rmetdet
!dir$ attributes align:64 :: Dvv
!dir$ attributes align:64 :: vvtemp
!dir$ attributes align:64 :: div_orig
!dir$ attributes align:64 :: div_test
```

Before loops:

DIR$ VECTOR ALWAYS ALIGNED

Compile code with:

-align array64byte

Dinv accesses are not contiguous

--how to fix:

```
real*8, Dimension(:,:,:,:), Allocatable :: Dinv
real*8, Dimension(:,:,:,:), Allocatable :: DinvC
```

```
allocate(Dinv(2,2,np,np))
allocate(DinvC(np,np,2,2))
```

```
deallocate (Dinv)
deallocate (DinvC)
```

```
 do l=1,2
  do k=1,2
   do j=1,np
    do i=1,np
     Dinv(k,l,i,j) = real((1-1)*np*np*2+(k-1)*np*np+(j-1)*np+i)*2.0d0
     DinvC(i,j,k,l) = Dinv(k,l,i,j)
   end do
  end do
 end do
end do
```

```
    ! convert to contra variant form and multiply by g
    do j=1,tripCount
!DIR$ VECTOR ALWAYS ALIGNED
      do i=1,tripCount
        !gv(i,j,1)=metdet(i,j)*(Dinv(1,1,i,j)*v(i,j,1) + Dinv(1,2,i,j)*v(i,j,2))
        !gv(i,j,2)=metdet(i,j)*(Dinv(2,1,i,j)*v(i,j,1) + Dinv(2,2,i,j)*v(i,j,2))
        gv(i,j,1)=metdet(i,j)*(DinvC(i,j,1,1)*v(i,j,1) + DinvC(i,j,1,2)*v(i,j,2))
        gv(i,j,2)=metdet(i,j)*(DinvC(i,j,2,1)*v(i,j,1) + DinvC(i,j,2,2)*v(i,j,2))
       enddo
    enddo
```

Second set of loops are more difficult

Vector Advisor and opt report shows loop vectorized but there are dependencies:

- dudx00 & dvdy00

gv(j,i,2) not a stride of 1

```
! compute d/dx and d/dy
do j=1,tripCount
   do k=1,tripCount
      dudx00=0.0d0
      dvdy00=0.0d0
      do i=1,tripCount
         dudx00 = dudx00 + Dvv(i,k  )*gv(i,j  ,1)
         dvdy00 = dvdy00 + Dvv(i,k  )*gv(j  ,i,2)
      end do
      div(k  ,j  ) = dudx00
      vvtemp(j  ,k  ) = dvdy00
   end do
end do
```

Loop 2 Remove dependency

Lab2_3

```
! compute d/dx and d/dy
do j=1,tripCount
   do k=1,tripCount
      dudx00=0.0d0
      dvdy00=0.0d0
      do i=1,tripCount
         dudx00 = dudx00 + Dvv(i,k  )*gv(i,j  ,1)
         dvdy00 = dvdy00 + Dvv(i,k  )*gv(j  ,i,2)
      end do
      div(k  ,j  ) = dudx00
      vvtemp(j  ,k  ) = dvdy00
   end do
end do
```

Remove dudx00

- Replace with div(l,j)

Remove dvdy00

- Replace with vvtemp(j,l)

need to initizialize both to 0 at the beginning

- Vvtemp=0.0d0
- Div = 0.0d0

But, still have gv(j,i,2) not being accessed contiguously

```
vvtemp=0.0d0
div=0.0d0
do j=1,np
    do k=1,np
      do i=1,np
        div(k,j) = div(k,j) + Dvv(i,k  )*gv(i,j  ,1)
        vvtemp(j ,k) = vvtemp(j,k) + Dvv(i,k  )*gv(j,i,2)
      end do
    end do
  end do
```

Added two Extra loops
- Performance will get worse
- But now able to vectorize further

Observe the pattern of vtemp calculation to transform it

Dvv(i,k)

row →

col ↓

10	50	90	130
20	60	100	140
30	70	110	150
40	80	120	160

Gv(j,i,n)

row →

col ↓

1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
4	8	12	16	20	24	28	32

n=1 n=2

Tip: Don't focus too much on the actual letters used, instead focus on how the multi-dimensional array is accessed

j,k,i: gv(j,i,2) == i,k,j: gv(i,j,2)

vtemp(j,k) row →

j,k,**i**; vvtemp(j,k) = vvtemp(j,k) + Dvv(**i**,k) * **gv(j,i,2)**

col ↓

10*17+20*21+30*25+40*29			

i = **1 -> tripCount**
k = **1**
j = 1

vvtemp(1,**1**) = vvtemp(1,**1**) + Dvv(**1**,**1**)*gv(1,**1**,2) + Dvv(**2**,**1**)*gv(1,**2**,2) + ...
= 0 + 10 * 17 + 20 * 21 + ...

Observe the pattern of vtemp calculation to transform it

Dvv(i,k)

row →

col ↓

10	50	90	130
20	60	100	140
30	70	110	150
40	80	120	160

Gv(j,i,n)

row →

col ↓

1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
4	8	12	16	20	24	28	32

n=1 n=2

vtemp(j,k) row →

j,**k**,i; vvtemp(j,**k**) = vvtemp(j,**k**) + Dvv(i,**k**) * **gv(j,i,2)**

col ↓

10*17+20*21+30*25+40*29	50*17+60*21+70*25+80*29		

i = **1 -> tripCount**
k = **2**
j = 1

vvtemp(1,**2**) = vvtemp(1,**2**) + Dvv(**1**,**2**)*gv(1,**1**,2) + Dvv(**2**,**2**)*gv(1,**2**,2) + ...
= 0 + 50 * 17 + 60 * 21 + ...

Observe the pattern of vtemp calculation to transform it

Dvv(i,k)

col ↓ / row →

10	50	90	130
20	60	100	140
30	70	110	150
40	80	120	160

Gv(j,i,n)

col ↓ / row →

1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
4	8	12	16	20	24	28	32
n=1				n=2			

vtemp(j,k)

col ↓ / row →

10*17+20*21+30*25+40*29	50*17+60*21+70*25+80*29	90*17+100*21+110*25+120*29	130*17+140*21+150*25+160*29
10*18+20*22+30*26+40*30	50*18+60*22+70*26+80*30	90*18+100*22+110*26+120*30	130*18+140*22+150*26+160*30
10*19+20*23+30*27+40*31	50*19+60*23+70*27+80*31	90*19+100*23+110*27+120*31	130*19+140*23+150*27+160*31
10*20+20*24+30*28+40*32	50*20+60*24+70*28+80*32	90*20+100*24+110*28+120*32	130*20+140*24+150*28+160*32

j,k,i; vvtemp(j,k) = vvtemp(j,k) + Dvv(i,k) * **gv(j,i,2)**

Observe the pattern of vtemp calculation to transform it

Dvv(i,k)

row →

col ↓

10	50	90	130
20	60	100	140
30	70	110	150
40	80	120	160

Gv(j,i,n)

row →

col ↓

1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
4	8	12	16	20	24	28	32

n=1 n=2

What if we did this instead:
Multiply down the columns and do partial sums

vtemp(j,k) row →

j,k,i; vvtemp(j,k) = vvtemp(j,k) + Dvv(i,k) * **gv(j,i,2)**

col ↓

10*17			
10*18			
10*19			
10*20			

vvtemp(1,1) = vvtemp(1,1) + Dvv(1,1)*gv(1,1,2) = 10*17
vvtemp(2,1) = vvtemp(2,1) + Dvv(1,1)*gv(2,1,2) = 10*18
vvtemp(3,1) = vvtemp(3,1) + Dvv(1,1)*gv(3,1,2) = 10*19
…

vvtemp(**i**,?) = vvtemp(**i**,?) + Dvv(?,?)*gv(**i**,?,2)

i = 1 -> tripCount

Observe the pattern of vtemp calculation to transform it

Dvv(i,k)

row →

col ↓

10	50	90	130
20	60	100	140
30	70	110	150
40	80	120	160

Gv(j,i,n)

row →

col ↓

1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
4	8	12	16	20	24	28	32

n=1 n=2

What if we did this instead:
Multiply down the columns and do partial sums

vtemp(j,k) row →

j,k,i; vvtemp(j,k) = vvtemp(j,k) + Dvv(i,k) * **gv(j,i,2)**

col ↓

10*17+20*21			
10*18+20*22			
10*19+20*23			
10*20+20*24			

vvtemp(1,1) = vvtemp(1,1) + Dvv(1,1)*gv(1,1,2) = 10*17
vvtemp(2,1) = vvtemp(2,1) + Dvv(1,1)*gv(2,1,2) = 10*18
vvtemp(3,1) = vvtemp(3,1) + Dvv(1,1)*gv(3,1,2) = 10*19
...

vvtemp(**i**,?) = vvtemp(**i**,?) + Dvv(?,?)*gv(**i**,?,2)

i = 1 -> tripCount

vvtemp(1,1) = vvtemp(1,1) + Dvv(**2**,1)*gv(1,**2**,2) = 20*21
vvtemp(2,1) = vvtemp(2,1) + Dvv(**2**,1)*gv(2,**2**,2) = 20*22
vvtemp(3,1) = vvtemp(3,1) + Dvv(**2**,1)*gv(3,**2**,2) = 20*23
...

...

vvtemp(i,?) = vvtemp(i,?) + Dvv(**k**,?)*gv(i,**k**,2)

i = 1 -> tripCount

k = 2

? Must then be j

Observe the pattern of vtemp calculation to transform it

Dvv(i,k) row →

col ↓

10	50	90	130
20	60	100	140
30	70	110	150
40	80	120	160

Gv(j,i,n) row →

col ↓

1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31
4	8	12	16	20	24	28	32

n=1 n=2

vtemp(j,k) row →

j,k,i; vvtemp(j,k) = vvtemp(j,k) + Dvv(i,k) * **gv(j,i,2)**

col ↓

10*17+20*21+30*25+40*29	50*17+60*21+70*25+80*29	90*17+100*21+110*25+120*29	130*17+140*21+150*25+160*29
10*18+20*22+30*26+40*30	50*18+60*22+70*26+80*30	90*18+100*22+110*26+120*30	130*18+140*22+150*26+160*30
10*19+20*23+30*27+40*31	50*19+60*23+70*27+80*31	90*19+100*23+110*27+120*31	130*19+140*23+150*27+160*31
10*20+20*24+30*28+40*32	50*20+60*24+70*28+80*32	90*20+100*24+110*28+120*32	130*20+140*24+150*28+160*32

j,k,i; vvtemp(j,k) = vvtemp(j,k) + Dvv(i,k) * **gv(j,i,2)**

↓

j,k,i; vvtemp(i,j) = vvtemp(i,j) + Dvv(k,j) * gv(i,k,2)

New loop transform now accesses
The data with stride of 1

Loop2 contiguous; but does not use zmm width

Not able to store data using inner loop dimension

Need to modify loop

j,k,i: div(k,j) = div(k,j) + Dvv(i,k)*gv(i,j,1)

j,i,k: div(k,j) = div(k,j) + DvvC(k,i)*gv(i,j,1)

DvvC = transpose(Dvv)

```
vvtemp=0.0d0
div=0.0d0
do j=1,np
      do k=1,np
        do i=1,np
          div(k,j) = div(k,j) + Dvv(i,k  )*gv(i,j  ,1)
            vvtemp(i,j) = vvtemp(i,j) + Dvv(k,j) * gv(i,k,2)
        end do
       end do
    end do
```

All Loops Vectorized

More optimizations:

- Put div=0.0d0 and vvtemp=0.0d0 in first loop
- Combine inner loop of bottom loop with 2nd block of loops
- Change constant tripCount to np (parameter or define)

Lab2_5

```
module subroutine test(div)
    real*8, dimension (np,np), intent(inout) :: div

    ! convert to contra variant form and multiply by g
    do j=1,tripCount
!DIR$ VECTOR ALWAYS ALIGNED
      do i=1,tripCount
        gv(i,j,1)=metdet(i,j)*(DinvC(i,j,1,1)*v(i,j,1) + DinvC(i,j,1,2)*v(i,j,2))
        gv(i,j,2)=metdet(i,j)*(DinvC(i,j,2,1)*v(i,j,1) + DinvC(i,j,2,2)*v(i,j,2))
      enddo
    enddo

    div=0.0d0
    vvtemp=0.0d0
    ! compute d/dx and d/dy
    do j=1,tripCount
       do i=1,tripCount
!DIR$ VECTOR ALWAYS ALIGNED
          do k=1,tripCount
             div(k,j) = div(k,j) + DvvC(k,i  )*gv(i,j  ,1)
          end do
       end do
       do k=1,tripCount
!DIR$ VECTOR ALWAYS ALIGNED
          do i=1,tripCount
             vvtemp(i,j) = vvtemp(i,j) + Dvv(k,j)*gv(i,k,2)
          end do
       end do
    end do

    do j=1,tripCount
!DIR$ VECTOR ALWAYS ALIGNED
       do i=1,tripCount
          div(i,j)=(div(i,j)+vvtemp(i,j))*(rmetdet(i,j)*rrearth)
       end do
    end do
end subroutine test
```

Solution

Optimized code has large speedup vs. original (~10x)

- 11.9 sec vs 1.18 sec

Code now vectorized, compiler knows trip count at compile time and can optimize further

Lab2_6

```
module subroutine test(div)
    real*8, dimension (np,np), intent(inout) :: div

    ! convert to contra variant form and multiply by g
    do j=1,np
!DIR$ VECTOR ALWAYS ALIGNED
      do i=1,np
        gv(i,j,1)=metdet(i,j)*(DinvC(i,j,1,1)*v(i,j,1) + DinvC(i,j,1,2)*v(i,j,2))
        gv(i,j,2)=metdet(i,j)*(DinvC(i,j,2,1)*v(i,j,1) + DinvC(i,j,2,2)*v(i,j,2))
        div(i,j)=0.0d0
        vvtemp(i,j)=0.0d0
      enddo
    enddo

    ! compute d/dx and d/dy
    do j=1,np
       do i=1,np
!DIR$ VECTOR ALWAYS ALIGNED
          do k=1,np
             div(k,j) = div(k,j) + DvvC(k,i  )*gv(i,j  ,1)
          end do
       end do
       do k=1,np
!DIR$ VECTOR ALWAYS ALIGNED
          do i=1,np
             vvtemp(i,j) = vvtemp(i,j) + Dvv(k,j)*gv(i,k,2)
          end do
       end do
!DIR$ VECTOR ALWAYS ALIGNED
       do i=1,np
          div(i,j)=(div(i,j)+vvtemp(i,j))*(rmetdet(i,j)*rrearth)
       end do
    end do
end subroutine test
```

Vectorization Review

Five main tools/resources

- Compiler options
- Compiler optimization reports
- Compiler generated annotated assembly
- Vector Advisor
- Intel developer's online documentation on vectorization

Notes

- Compiler will always be conservative, if dependency will not vectorize unless explicitly specified by user.
- AVX-512 has new vectorization capabilities for branches (k-mask vectors)
- Alignment + access pattern optimizations + Loop transformations are common vectorization tricks

MEMORY BANDWIDTH

Memory bandwidth lab learning goals

Understanding the behavior and performance implications of memory modes

Usage of Intel® Software Development Tools to identify software bottlenecks

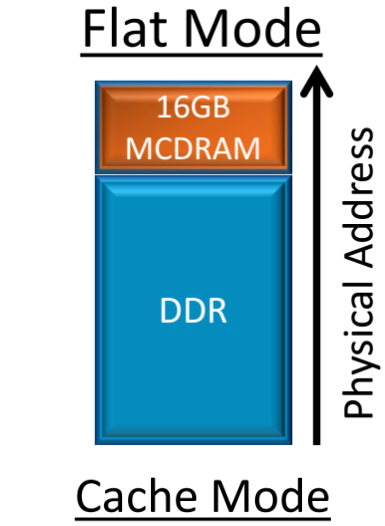
Memory modes

Flat mode

- MCDRAM mapped to physical address space
- MCDRAM exposed as a NUMA node
- Accessed through *numactl* or Memkind library

Cache mode

- MCDRAM acts as a memory-side cache
- No source changes are needed to use MCDRAM cache
- MCDRAM cache misses are expensive



Problem sizing

Strong Scaling

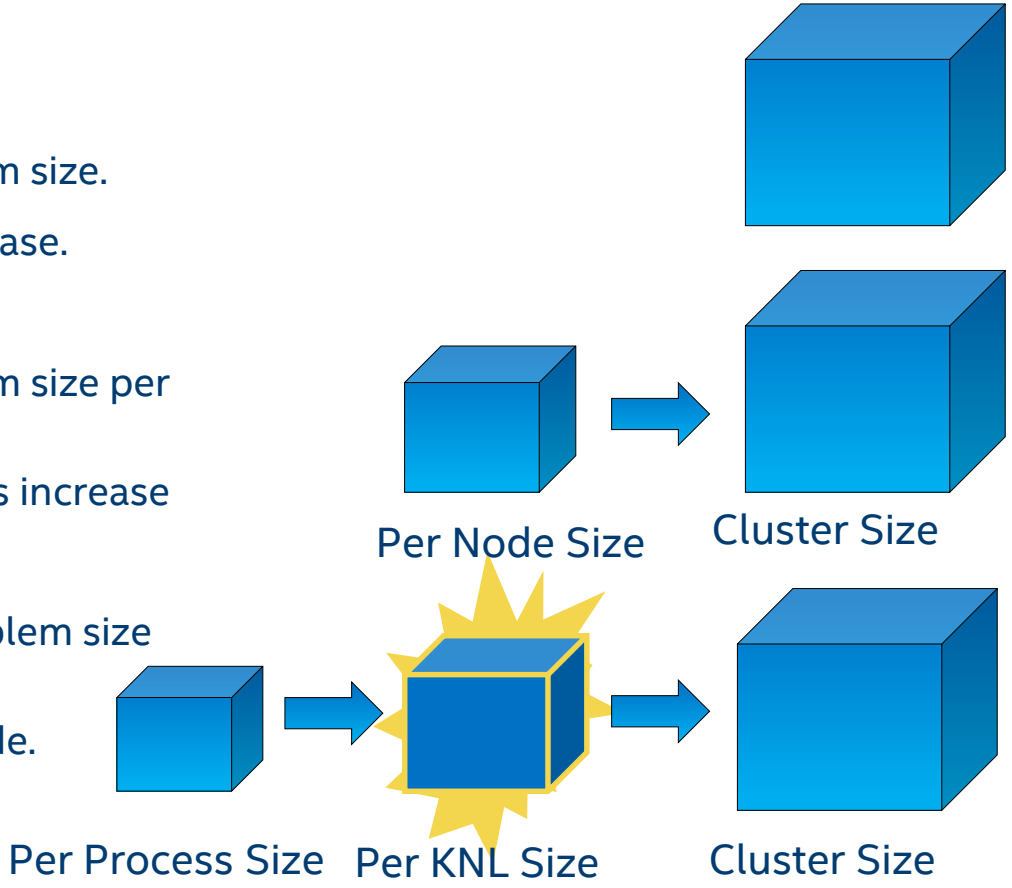
- Variable number of ranks. Fixed problem size.
- Goal: Decrease time as processors increase.

Weak Scaling

- Variable number of ranks. Fixed problem size per process.
- Goal: Increase throughput as processors increase

Problem Sizing

- Variable number of ranks. Variable problem size per process.
- Goal: Maximize throughput per KNL node.
- This is critical for KNL.



Lab overview

MiniGhost Demo

Run code with different problem sizes in flat mode

Run code with different problem sizes in cache mode

Run under various tools to find performance bottlenecks

Resolve bottlenecks, repeat until satisfied

Identifying memory mode using *numactl*

Cache mode

- MCDRAM behaves like a memory side cache and is not visible to software.
- App will use MCDRAM cache by default

```
$numactl -H

node 0 size: 98200 MB
node 0 free: 93215 MB
node distances:
node   0
  0:  10
```

Identifying memory mode using *numactl*

Flat mode

```
$numactl -H

node 0 size: 98200 MB
node 0 free: 92651 MB
node 1 cpus:
node 1 size: 16384 MB
node 1 free: 15568 MB
node distances:
node   0   1
  0:  10  31
  1:  31  10
```

MCDRAM only node. No CPUs

MCDRAM is farther. DDR will be default

Running app using *numactl*

- Bind to MCDRAM
  - Will fail when sufficient memory is not available
  - Accepts multiple nodes as arguments

```
$numactl -m 1 ./app
```

- Prefer MCDRAM
  - Will fall back to other nodes when sufficient memory is not available
  - Accepts a single numa node as argument

```
$numactl -p 1 ./app
```

Running MiniGhost in flat mode

Running different problem sizes in flat mode

- Try varying the problem size keeping the number of ranks constant.
- Observe performance for:
  - DDR only
  - MCDRAM using *numactl* bind
  - MCDRAM using *numactl* preferred
- Observe memory footprint using *numastat*
  - `$watch –n1 numastat –m`

```
Per-node system memory usage (in MBs):
                          Node 0          Node 1           Total
                 --------------- --------------- ---------------
MemTotal               98200.81        16384.00       114584.81
MemFree                92530.91        15568.88       108099.78
MemUsed                 5669.91          815.12         6485.03
Active                    68.22            0.00           68.22
Inactive                 169.52            0.00          169.52
Active(anon)              48.59            0.00           48.59
Inactive(anon)            16.97            0.00           16.97
Active(file)              19.63            0.00           19.63
Inactive(file)           152.54            0.00          152.54
```

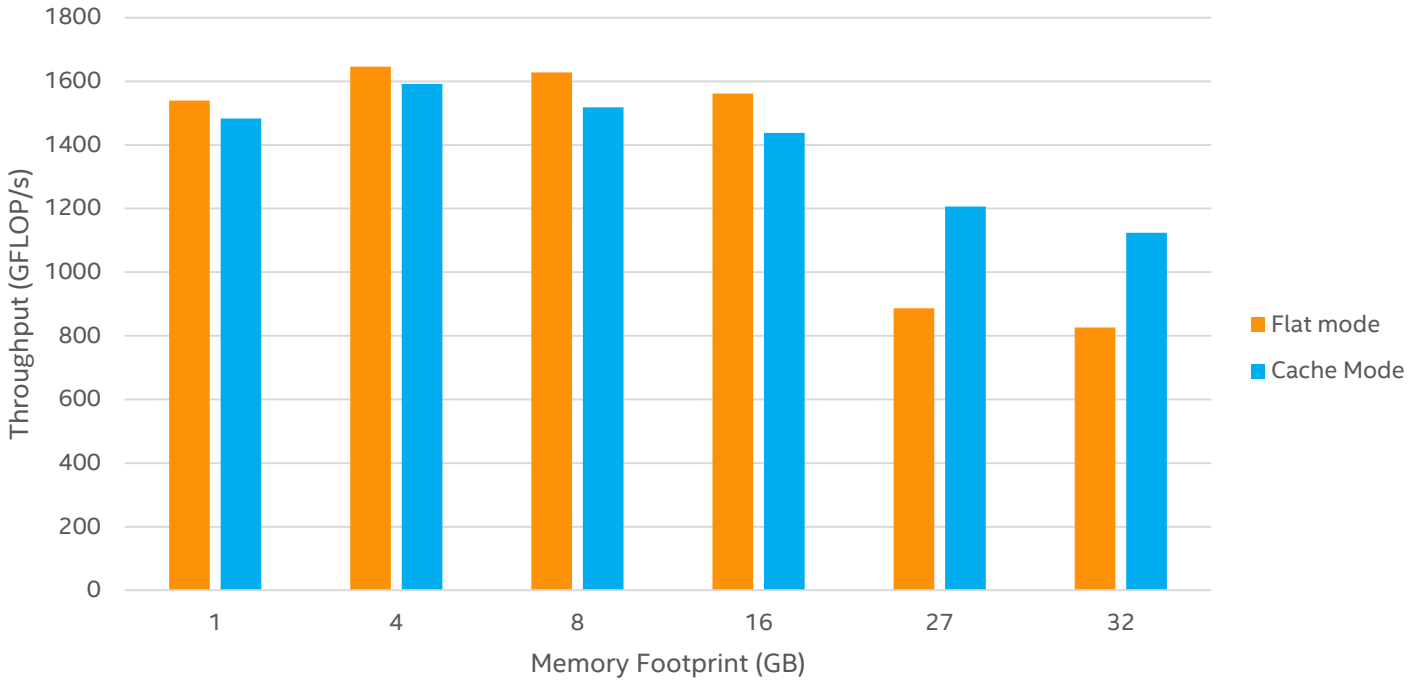
Keep an eye on MemUsed per numa node

Running MiniGhost in cache mode

Running different problem sizes in cache mode

- Try varying the problem size keeping the number of ranks constant.
- Observe performance and compare against flat mode.

Flat mode vs cache mode

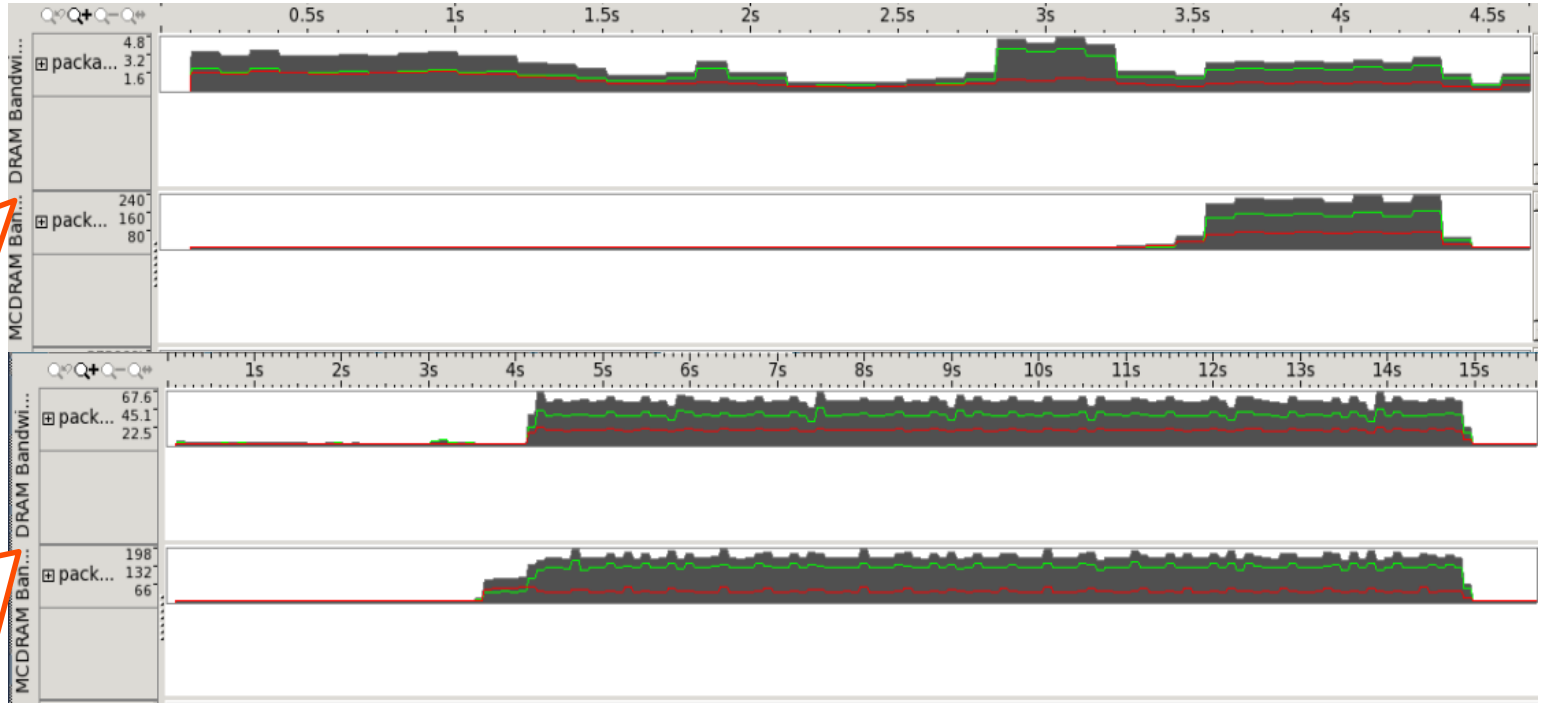


Memory access analysis in flat mode

Take a look at the pre-collected Intel® VTune Amplifier™ XE memory access analysis results for different problem sizes:

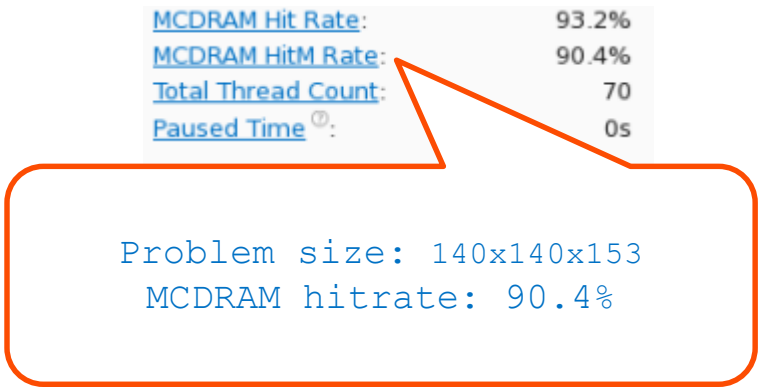
Problem size: 140x140x153 Peak BW: 240 GB/s

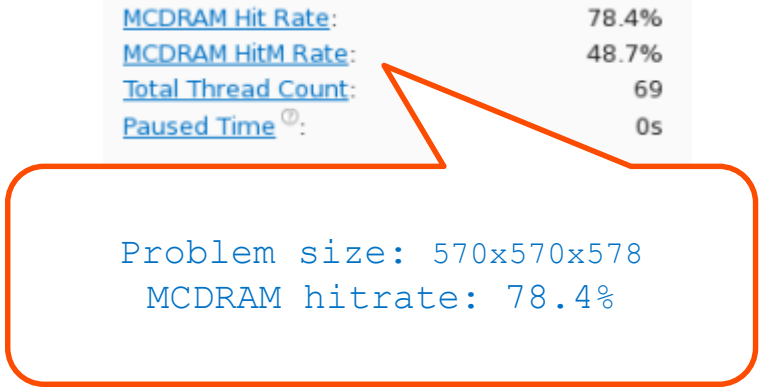
Problem Size: 336x336x340 Peak BW: 198 GB/s + 67.6 GB/s



Memory access analysis in cache mode (1/2)

Take a look at the pre-collected Intel® VTune Amplifier™ XE memory access analysis results for different problem sizes:





Memory access analysis in cache mode (2/2)

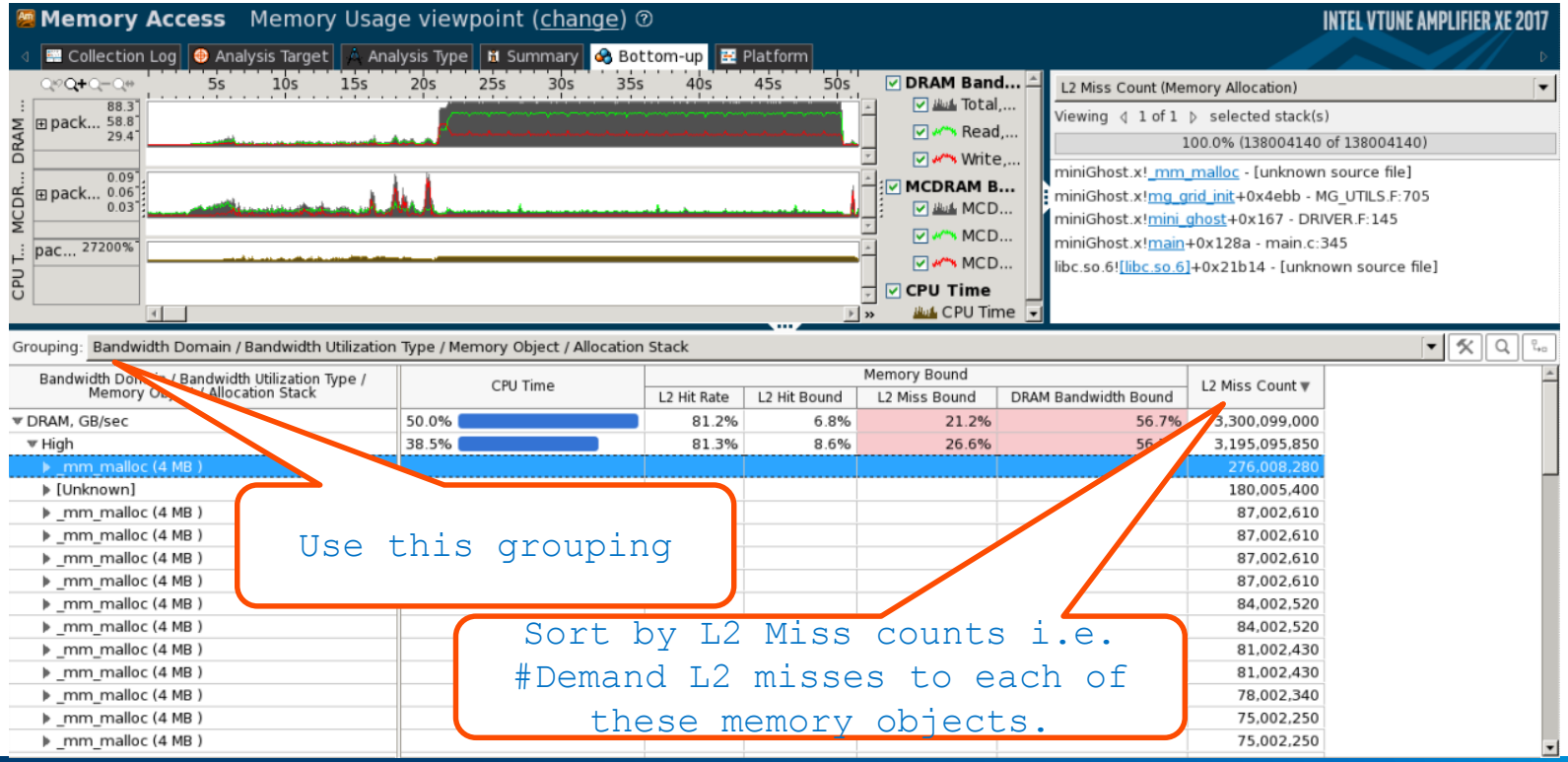
Intel® VTune Amplifier™ XE memory access analysis results for different problem sizes:

Problem size: 140x140x153
Peak BW: 223 GB/s + 12.1 GB/s

Problem Size: 570x570x578
Peak BW: 173 GB/s + 88.2 GB/s

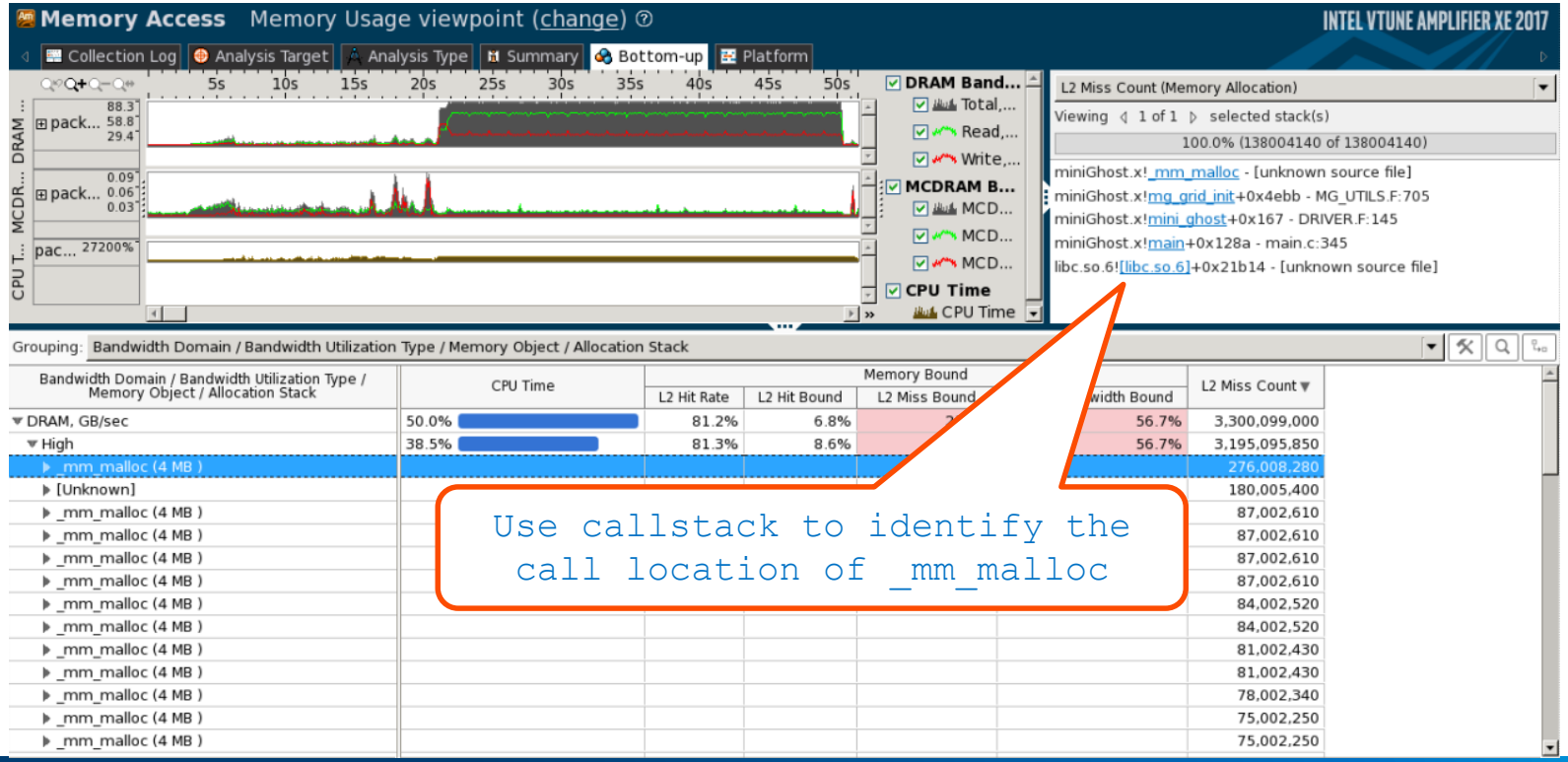
Identifying high bandwidth memory objects (1/3)

Memory object analysis: DDR only



Identifying high bandwidth memory objects (2/3)

Memory object analysis: DDR only



Identifying high bandwidth memory objects (3/3)

MG_UTILS.F

```
685           CALL MG_INIT_GRID ( GRID38, IERR )
686        END IF
687
688        IF ( NUM_VARS > 38 ) THEN
689           ALLOCATE ( GRID39( 0:NX+1, 0:NY+1, 0:NZ+1 ), STAT = IERR )
690           CALL MG_ASSERT ( IERR, 'GRID_INIT: ALLOCATE ( GRID39 )', (NX+2)*(NY+2)*(NZ+2) )
691           CALL MG_INIT_GRID ( GRID39, IERR )
692        END IF
693
694        IF ( NUM_VARS > 39 ) THEN
695           ALLOCATE ( GRID40( 0:NX+1, 0:NY+1, 0:NZ+1 ), STAT = IERR )
696           CALL MG_ASSERT ( IERR, 'GRID_INIT: ALLOCATE ( GRID40 )', (NX+2)*(NY+2)*(NZ+2) )
697           CALL MG_INIT_GRID ( GRID40, IERR )
698        END IF
699
700        IF ( NUM_VARS > 40 ) THEN
701           IERR = -1
702           CALL MG_ASSERT ( IERR, 'GRID_INIT: TOO MANY VARS', NUM_VARS )
703        END IF
704
705        ALLOCATE ( WORK( 0:NX+1, 0:NY+1, 0:NZ+1 ), STAT = IERR )
706        CALL MG_ASSERT ( IERR, 'GRID_INIT: ALLOCATE ( WORK )', (NX+2)*(NY+2)*(NZ+2) )
707
708        RETURN
709
710     END SUBROUTINE MG_GRID_INIT
711
712  !  ================================================================================
713
```

High BW memory object identified is work

Using FASTMEM directive to place high bandwidth memory objects in MCDRAM

MG_CONSTANTS.F

```
120    REAL(KIND=MG_REAL),    DIMENSION(:), ALLOCATABLE ::    &
121
122       FLUX_OUT,                        &  ! Keeps track of heat dissapation out of physical domain.
123       SOURCE_TOTAL                        ! Keeps track of heat inserted into each variable.
124
125    REAL(KIND=MG_REAL),    DIMENSION(:,:), ALLOCATABLE ::    &
126       SPIKES                              ! Heat spike to be inserted.
127
128 !DIR$ ATTRIBUTES FASTMEM :: WORK
129    REAL(KIND=MG_REAL),    DIMENSION(:,:,:),  ALLOCATABLE ::    &
130       GRID1,  GRID2,  GRID3,  GRID4,  GRID5,  GRID6,  GRID7,  GRID8,  GRID9,  GRID10,     & !
131       GRID11, GRID12, GRID13, GRID14, GRID15, GRID16, GRID17, GRID18, GRID19, GRID20,     & !
132       GRID21, GRID22, GRID23, GRID24, GRID25, GRID26, GRID27, GRID28, GRID29, GRID30,     & !
133       GRID31, GRID32, GRID33, GRID34, GRID35, GRID36, GRID37, GRID38, GRID39, GRID40,     & !
134       WORK
135
136    INTEGER(KIND=MG_INT), PARAMETER :: &
137
138       ROOT_PE = 0,                     &
139
```

Allocate WORK in MCDRAM

Next Steps

Identify other high bandwidth memory objects.

Use FASTMEM to place memory objects in MCDRAM

Compare performance between *numactl* and FASTMEM

SUMMARY

Performance is Becoming More Complex

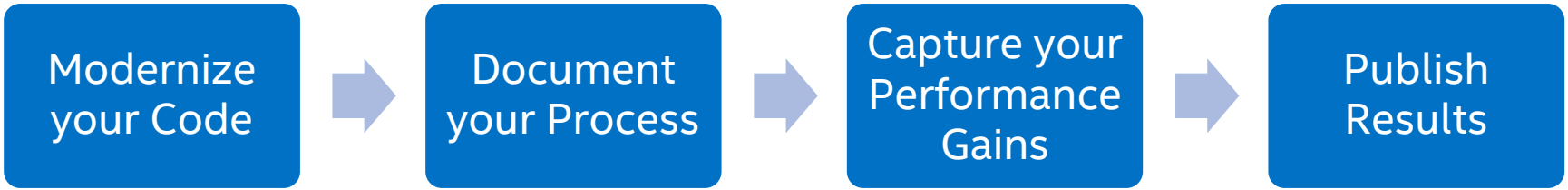
More cores

More memory options

New vectorization capabilities

Some previous techniques must evolve to fully utilize new capabilities

Challenge

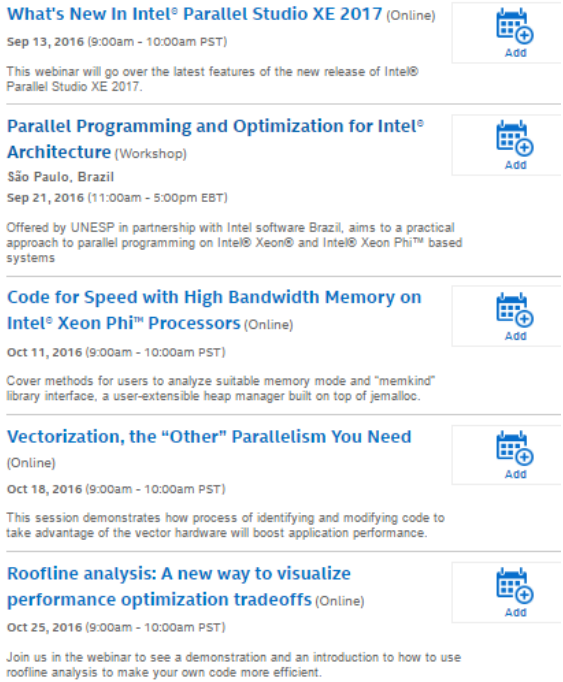

Modernize your Code
Document your Process
Capture your Performance Gains
Publish Results

ADDITIONAL RESOURCES

Educating with Webinar Series about 2017 Tools

- Expert talks about the new features
- Series of live webinars, September 13 – November 8, 2016
- Attend live or watch after the fact.

https://software.intel.com/events/hpc-webinars



Educating with High-Performance Programming Book

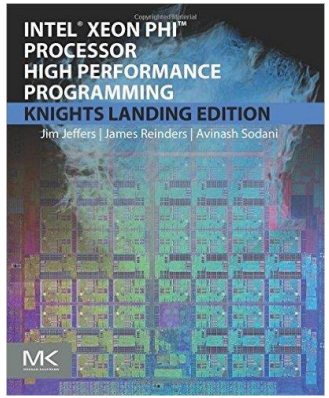
Knights-Landing-specific details, programming advice, and real-world examples.

Intel® Xeon Phi™ Processor High Performance Programming

- Techniques to generally increase program performance on any system and prepare you better for Intel Xeon Phi processors.

Available as of June 2016

http://lotsofcores.com



" I believe you will find this book is an invaluable reference to help develop your own Unfair Advantage."

James A.
Manager
Sandia National Laboratories

More Education with software.intel.com/moderncode

- Online community growing collection of tools, trainings, support
  - Features Black Belts in parallelism from Intel and the industry
- Intel® HPC Developer Conferences developers share proven techniques and best practices
  - hpcdevcon.intel.com

- Hands-on training for developers and partners with remote access to Intel® Xeon® processor and Xeon Phi™ coprocessor-based clusters.
  - software.intel.com/icmp
- Developer Access Program provides early access to Intel® Xeon Phi™ processor codenamed Knights Landing plus one-year license for Intel® Parallel Studio XE Cluster Edition.
  - http://dap.xeonphi.com/

Choices to Fit Needs: Intel® Tools

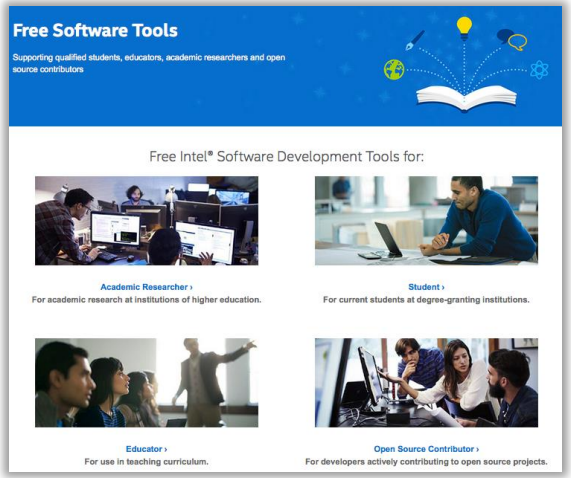
All Products with support – worldwide, for purchase.

- Intel® Premier Support - private direct support from Intel
- support for past versions
- software.intel.com/products

Most Products without Premier support – via special programs for those who qualify

- students, educators, classroom use, open source developers, and academic researchers
- software.intel.com/qualify-for-free-software

- Intel® Performance Libraries without Premier support -Community licensing for Intel performance libraries
  - no royalties, no restrictions based on company or project size
  - software.intel.com/nest



Community support only – all tools:
Students, Educators, classroom use,
Open Source Developers,
Academic Researchers (qualification required)

Community support only – Intel Performance Libraries:
Community Licensing (no qualification required)

INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

THANK YOU FOR YOUR TIME

Sumedh Naik, James Tullos, Antonio Valles, Michael Hebenstreit, Mallick Arigapudi, Zakhar Matveev

www.intel.com/hpcdevcon

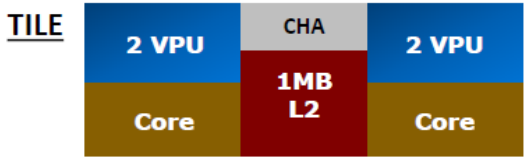
INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

BACK-UP

Running on KNL

Multiple memory modes and cluster modes

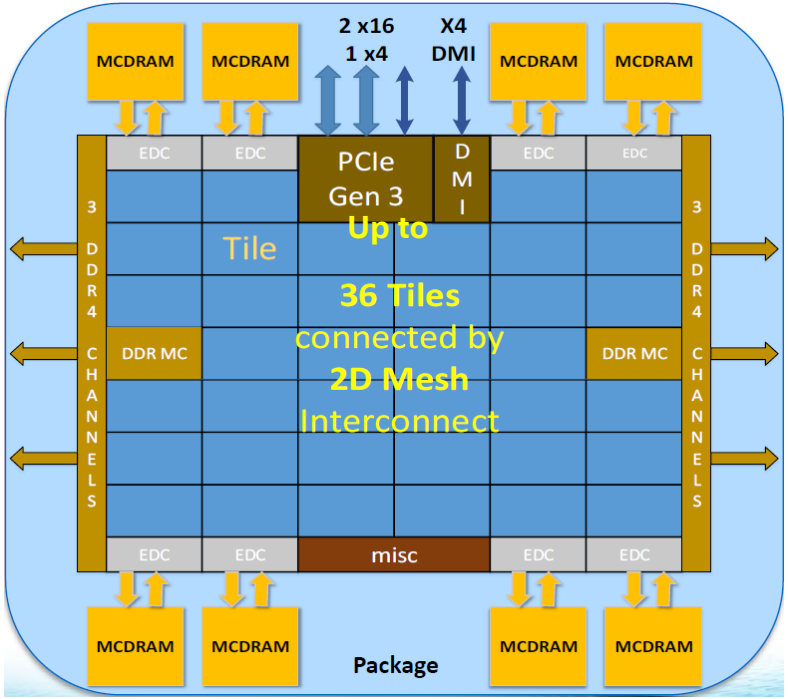


Memory Modes

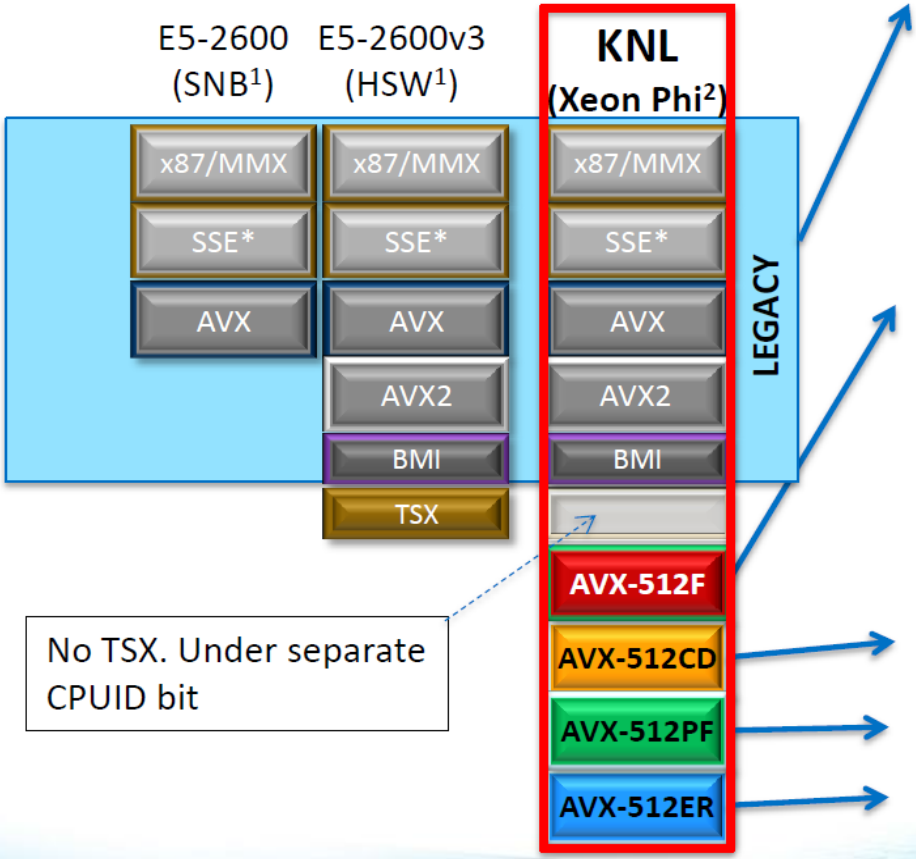
Flat Mode	Hybrid	Cache
DDR and MCDRAM as different NUMA nodes	Flat Mode + Cache	MCDRAM used as memory-side cache

Cluster Modes

All2All	Hemisphere/ Quadrant	SNC2/SNC4
No affinity between CHA and memory channel	Affinity between CHA and memory channel	Sub NUMA Cluster Affinitizes tile, CHA, and memory channel



KNL ISA



KNL implements all legacy instructions

- Legacy binary runs w/o recompilation
- KNC binary requires recompilation

KNL introduces AVX-512 Extensions

- 512-bit FP/Integer Vectors
- 32 registers, & 8 mask registers
- Gather/Scatter

Conflict Detection: Improves Vectorization

Prefetch: Gather and Scatter Prefetch

Exponential and Reciprocal Instructions

Previous Code name Intel® Xeon® processors
Xeon Phi = Intel® Xeon Phi™ processor

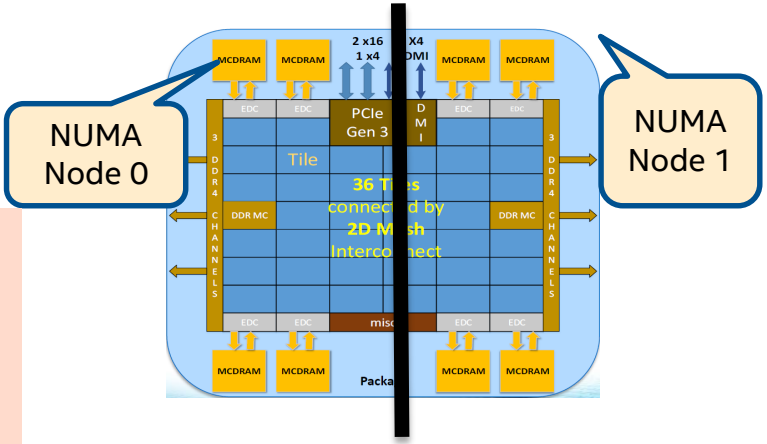
Source: Avinash Sodani's Hot chip's presentation: http://www.hotchips.org/wp-content/uploads/hc_archives/hc27/HC27.25-Tuesday-Epub/HC27.25.70-Processors-Epub/HC27.25.710-Knights-Landing-Sodani-Intel.pdf

Cache Mode / SNC-2

2 NUMA nodes present

- Symmetrical
  - Each has half CPUs, DDR, & MCDRAM
- Similar to a two-socket HSW
- MPI: use at least 2 MPI ranks

SNC Mode \ Memory Mode	Cache	Flat	Hybrid
No SNC			
SNC-2	2		
SNC-4			



numactl –H output **(on KNL)**

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 .. 236 237
node 0 size: 48154 MB
node 1 cpus: 34 35 .. 270 271
node 1 size: 48460 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

For an app running on node 0, node 1 is farther

numactl –H output **(on BDW-EP)**

```
available: 2 nodes (0-1)
node 0 cpus: 0 1 .. 52 53
node 0 size: 65439 MB
node 1 cpus: 18 19 .. 70 71
node 1 size: 65536 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
```

```
mpirun –n 64 ./app
```

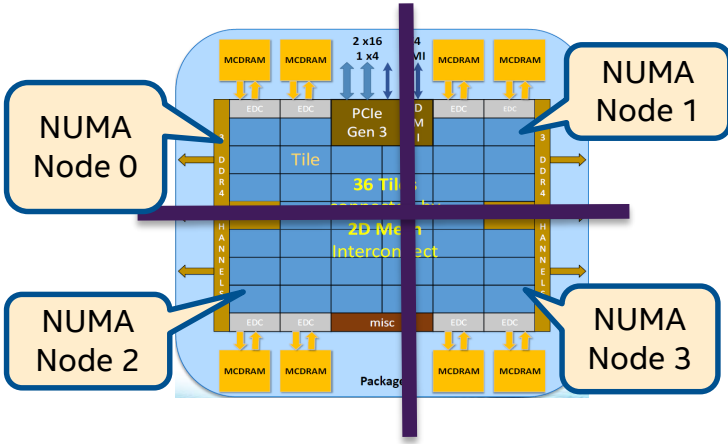
Cache Mode / SNC-4

4 NUMA nodes present

- Symmetrical
  - Each has quarter of DDR, MCDRAM & CPUs*
    - 68-core part has 18,18,16,16
- Similar to a four-socket HSW
- MPI: use at least 4 MPI ranks

Memory Mode

SNC Mode	Cache	Flat	Hybrid
No SNC			
SNC-2			
SNC-4	4		



numactl –H output (on KNL)

```
available: 4 nodes (0-3)
node 0 cpus: 0 1 .. 220 221
node 0 size: 23921 MB
node 1 cpus: 18 19 .. 238 239
node 1 size: 24231 MB
node 2 cpus: 36 37 .. 254 255
node 2 size: 24232 MB
node 3 cpus: 52 53 .. 270 271
node 3 size: 24229 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

For an app running on node 0, node 1-3 is farther

numactl –H output (on BDW-EX)

```
available: 4 nodes (0-3)
node 0 cpus: 0 1 .. 7 118 119
node 0 size: 130946 MB
node 1 cpus: 24 25 .. 142 143
node 1 size: 131072 MB
node 2 cpus: 48 49 .. 166 167
node 2 size: 131072 MB
node 3 cpus: 72 73 .. 190 191
node 3 size: 131072 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

```
mpirun –n 64 ./app
```

Flat Mode / SNC-2

Memory Mode

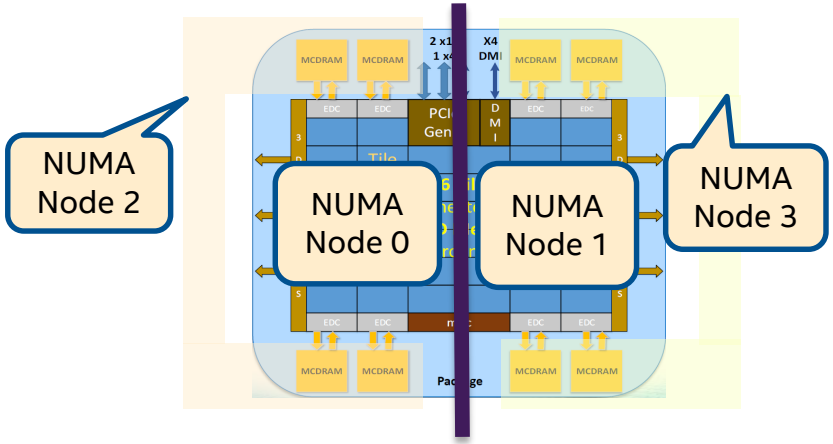
SNC Mode	Cache	Flat	Hybrid
No SNC			
SNC-2		4	
SNC-4			

4 NUMA nodes

- Asymmetrical
  - Node 0: ½ CPUs and ½ DDR
    - Node 1 same
  - Node 2: ½ MCDRAM
    - Node 3 same

```
numactl –H output
available: 4 nodes (0-3)
node 0 cpus: 0 1 .. 236 237
node 0 size: 48155 MB
node 1 cpus: 34 35 .. 270
271
node 1 size: 48464 MB
node 2 cpus:
node 2 size: 8078 MB
node 3 cpus:
node 3 size: 8076 MB
node distances:
node   0   1   2   3
  0:  10  21  31  41
  1:  21  10  41  31
  2:  31  41  10  41
  3:  41  31  41  10
```

For Node 0, closest MCDRAM Node is 2




```
mpirun numactl –m 2,3 ./mpi_app
mpirun –perhost 32  numactl --preferred 2 ./app : numactl --preferred 3 ./app
```

- I_MPI_HBW_POLICY will be available later to simplify the command line (to omit node numbers)

Flat Mode / SNC-4

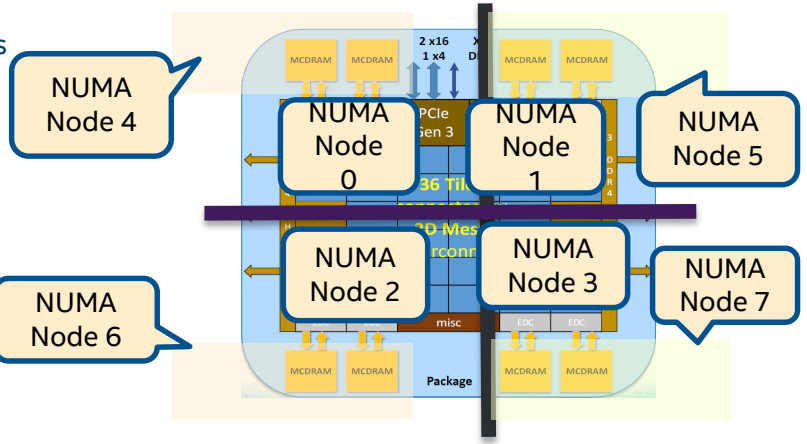
8 NUMA nodes

```
numactl -H output
available: 8 nodes (0-7)
node 0 cpus: 0 1 .. 220 221
node 0 size: 23922 MB
node 1 cpus: 18 19 .. 238 239
node 1 size: 24231 MB
node 2 cpus: 36 37 .. 254 255
node 2 size: 24232 MB
node 3 cpus: 52 53 .. 270 271
node 3 size: 24232 MB
node 4 cpus:
node 4 size: 4039 MB
node 5 cpus:
node 5 size: 4039 MB
node 6 cpus:
node 6 size: 4039 MB
node 7 cpus:
node 7 size: 4036 MB
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  21  21  21  31  41  41  41
  1:  21  10  21  21  41  31  41  41
  2:  21  21  10  21  41  41  31  41
  3:  21  21  21  10  41  41  41  31
  4:  31  41  41  41  10  41  41  41
  5:  41  31  41  41  41  10  41  41
  6:  41  41  31  41  41  41  10  41
  7:  41  41  41  31  41  41  41  10
```

- Asymmetrical
  - Node 0: ¼ CPUs and ¼ DDR
    - Node 1-3 same*
    - # CPUs can differ w/ 68 cores
  - Node 4: ¼ MCDRAM
    - Node 5,6,7 same

Memory Mode

SNC Mode	Cache	Flat	Hybrid
No SNC			
SNC-2			
SNC-4		8	




```
mpirun numactl -m 4,5,6,7 ./mpi_app
mpirun -perhost 16  numactl --preferred 5 ./app : numactl --preferred 6 ./app : ...
```

* More complicated command line needed to use all 68 cores in SNC-4 mode (see backup slide)

- MPI_HBW_POLICY will be available later to simplify the command line

Use Advisor to help with loop2 (lab2)

Summary | Survey Report | Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Max. Site Footprint	Site Name	Recommendations
[loop in test1 at lab2_3.F90:1 ...	No information available	60% / 20% / 20%	Mixed strides	504B	loop_site_19	

Memory Access Patterns Report | Dependencies Report | Recommendations

ID	Stride	Type	Source	Nested Function	Variable references	Access Footprint	Modules	Site Name
P1	8	Constant stride	lab2_3.F90:184			392B	lab2_3	loop_site_19
P2		Gather stride	lab2_3.F90:185			504B	lab2_3	loop_site_19
P3		Parallel site information	lab2_3.F90:184				lab2_3	loop_site_19
P5	0	Uniform stride	lab2_3.F90:184			8B	lab2_3	loop_site_19
P6	1	Unit stride	lab2_3.F90:184			448B	lab2_3	loop_site_19
P7	1	Unit stride	lab2_3.F90:185			56B	lab2_3	loop_site_19

Line	Source	Total Time	%	Loop Time	%	Traits
163	! - Change access of Dinv to be continuous					
164	! - Use VECTOR ALWAYS ALIGNED everywhere					
165	⊞subroutine test1(div)	0.040s				
166						
167	real*8, dimension (np,np), intent(inout) :: div					
168	! convert to contra variant form and multiply by g					
169	do j=1,np					
170	!DIR$ VECTOR ALWAYS ALIGNED					
171	do i=1,np					
172	gv(i,j,1)=metdet(i,j)*(DinvC(i,j,1,1)*v(i,j,1) + DinvC(i,j,1,2)*v(i,j,2))	1.040s				FMA
173	gv(i,j,2)=metdet(i,j)*(DinvC(i,j,2,1)*v(i,j,1) + DinvC(i,j,2,2)*v(i,j,2))	0.560s				FMA
174	enddo					
175	enddo					
176						
177	div=0.0d0	0.420s				
178	⊞ vvtemp=0.0d0	0.580s		0.500s		
179	! compute d/dx and d/dy					
180	do j=1,np					
181	do k=1,np					
182	!DIR$ VECTOR ALWAYS ALIGNED					
183	do i=1,np					
184	⊞ div(k,j) = div(k,j) + Dvv(i,k)*gv(i,j,1)	4.580s		7.700s		Extracts
185	vvtemp(j,k) = vvtemp(j,k) + Dvv(i,k)*gv(j,i,2)	3.640s				Extracts; Gathers; Mask Manipulations
186	end do					
187	⊞ end do					
188	end do					
189						
190	do j=1,np					
191	!DIR$ VECTOR ALWAYS ALIGNED					
192	do i=1,np					
193	⊞ div(i,j)=(div(i,j)+vvtemp(i,j))*(rmetdet(i,j)*rrearth)	0.280s				
194	end do	0.020s				
195	end do					
196						
197	end subroutine test1	0.080s				


intel®
experience
what's inside™