Overview of Performance Optimization on Intel® Xeon Phi™

Code Named Knights Landing (KNL)

Intel® Software Development Products

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

Focus Areas for Optimization

Optimization Focus Areas

Parallelism

Vectorization

Memory BW

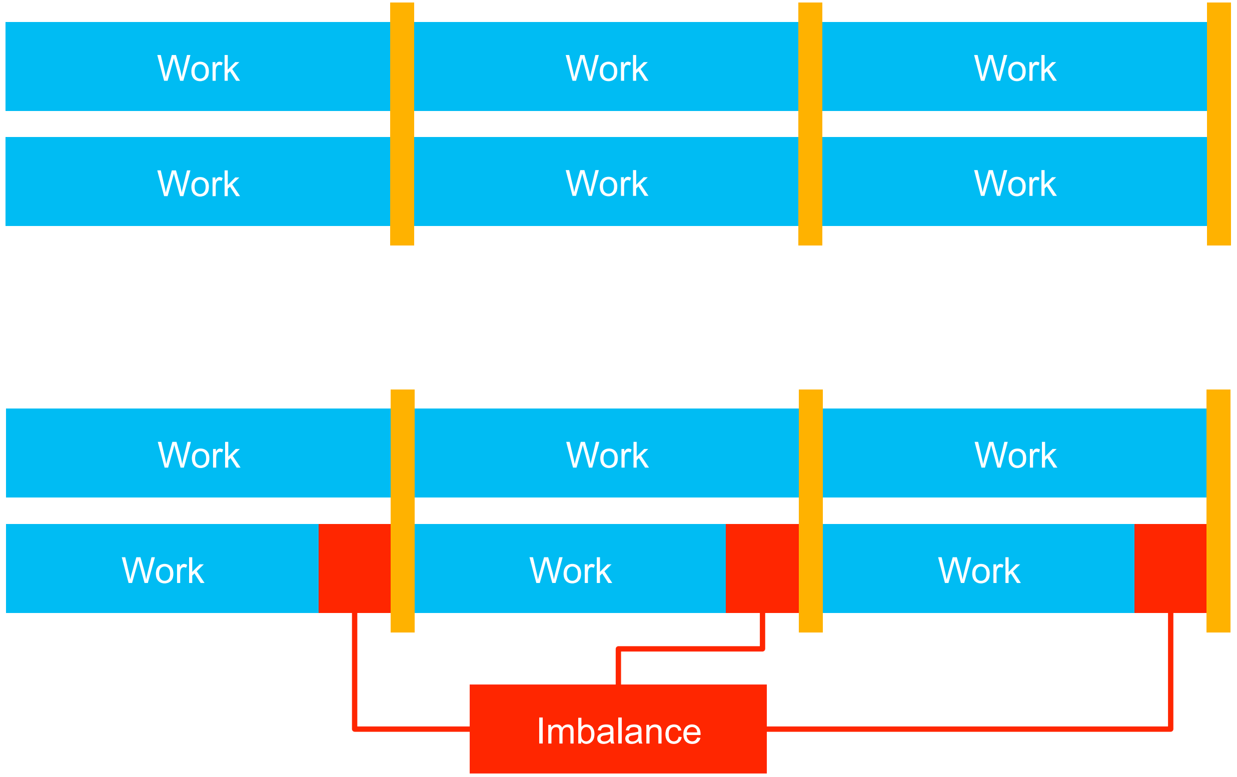
Parallelism on KNL

Multiple Threading Options

- Automatic Parallelism in Intel® Compilers
- OpenMP*
- Intel® Threading Building Blocks
- Threading inside of performance libraries

Also, MPI and MPI+Threading

Defining Imbalance in Parallelism



Vectorization on KNL

AVX-512 vector lanes

Automatic vectorization in compiler

- Sometimes needs help with directives/pragmas

SIMD loops: syntax

#pragma omp simd [*clauses*]

for-loop

!$omp simd [*clauses*]

do-loops

*[***!$omp end simd***]*

Loop has to be in “Canonical loop form”

- as do/for worksharing

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 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

private (list)

lastprivate (list)

reduction (operator:list)

collapse (n)

Same as existing clauses

SIMD functions: Syntax

#pragma omp declare simd ***[****clauses****]***

[#pragma omp declare simd [*clauses****]]***

function definition or declaration

!$omp declare simd (*function-or-procedure-name***)** ***[****clauses****]***

Instructs the compiler to

- generate a SIMD-enabled version(s) of a given function
- that a SIMD-enabled version of the function is available to use from a SIMD loop

SIMD functions: clauses

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

- function always called from inside an if statement

notinbranch

- function never called from inside an if statement

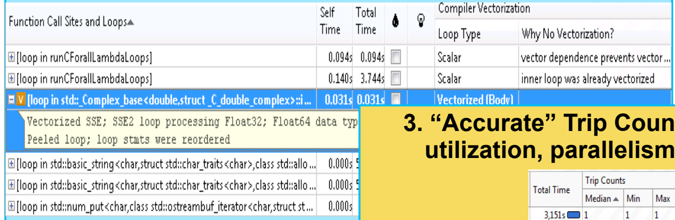
linear(*argument-list[:linear-step]*)

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

Same as before

5 Steps to Efficient Vectorization - Vector Advisor
(part of Intel® Advisor, Parallel Studio, Cluster Studio)

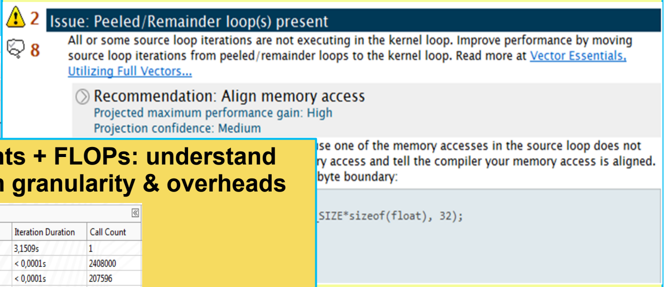
1. Compiler diagnostics + Performance Data + SIMD efficiency information



Function Call Sites and Loops	Self Time	Total Time			Compiler Vectorization	
					Loop Type	Why No Vectorization?
[loop in runCForallLambdaLoops]	0.094s	0.094s			Scalar	vector dependence prevents vector...
[loop in runCForallLambdaLoops]	0.140s	3.744s			Scalar	inner loop was already vectorized
V [loop in std::_Complex_base<double,struct _C_double_complex>::i...	0.031s	0.031s			Vectorized (Body)	
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	5				
[loop in std::basic_string<char,struct std::char_traits<char>,class std::allo...	0.000s	5				
[loop in std::num_put<char,class std::ostreambuf_iterator<char,struct st...	0.000s					

```
Vectorized SSE; SSE2 loop processing Float32; Float64 data typ
Peeled loop; loop stmts were reordered
```

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



2 Issue: Peeled/Remainder loop(s) present

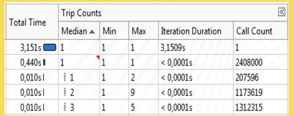
8 All or some source loop iterations are not executing in the kernel loop. Improve performance by moving source loop iterations from peeled/remainder loops to the kernel loop. Read more at Vector Essentials, Utilizing Full Vectors...

Recommendation: Align memory access
Projected maximum performance gain: High
Projection confidence: Medium

use one of the memory accesses in the source loop does not
ry access and tell the compiler your memory access is aligned.
byte boundary:

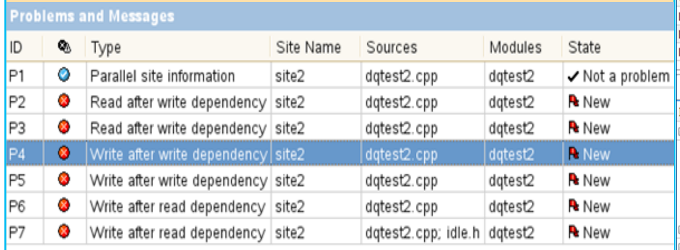
```
_SIZE*sizeof(float), 32);
```

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



Total Time	Trip Counts				
	Median	Min	Max	Iteration Duration	Call Count
3,151s	1	1	1	3,1509s	1
0,440s	1	1	1	< 0,0001s	2408000
0,010s	1	1	2	< 0,0001s	207596
0,010s	2	1	9	< 0,0001s	1173619
0,010s	3	1	5	< 0,0001s	1312315

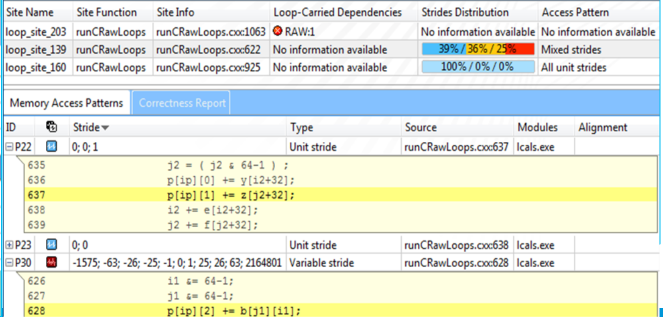
4. Loop-Carried Dependency Analysis



Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	site2	dqtest2.cpp	dqtest2	✓ Not a problem
P2		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P3		Read after write dependency	site2	dqtest2.cpp	dqtest2	New
P4		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P5		Write after write dependency	site2	dqtest2.cpp	dqtest2	New
P6		Write after read dependency	site2	dqtest2.cpp	dqtest2	New
P7		Write after read dependency	site2	dqtest2.cpp; idle.h	dqtest2	New

5. Memory Access Patterns Analysis



Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_203	runCRawLoops	runCRawLoops.cxx:1063	RAW:1	No information available	No information available
loop_site_139	runCRawLoops	runCRawLoops.cxx:622	No information available	39% / 36% / 25%	Mixed strides
loop_site_160	runCRawLoops	runCRawLoops.cxx:925	No information available	100% / 0% / 0%	All unit strides

Memory Access Patterns | Correctness Report

ID		Stride	Type	Source	Modules	Alignment
P22		0; 0; 1	Unit stride	runCRawLoops.cxx:637	lcals.exe	

```
j2 = ( j2 & 64-1 ) ;
p[ip][0] += y[i2+32];
p[ip][1] += z[j2+32];
i2 += e[i2+32];
j2 += f[j2+32];
```

ID		Stride	Type	Source	Modules	Alignment
P23		0; 0	Unit stride	runCRawLoops.cxx:638	lcals.exe	
P30		-1575; -63; -26; -25; -1; 0; 1; 25; 26; 63; 2164801	Variable stride	runCRawLoops.cxx:628	lcals.exe	

```
i1 &= 64-1;
j1 &= 64-1;
p[ip][2] += b[j1][i1];
```

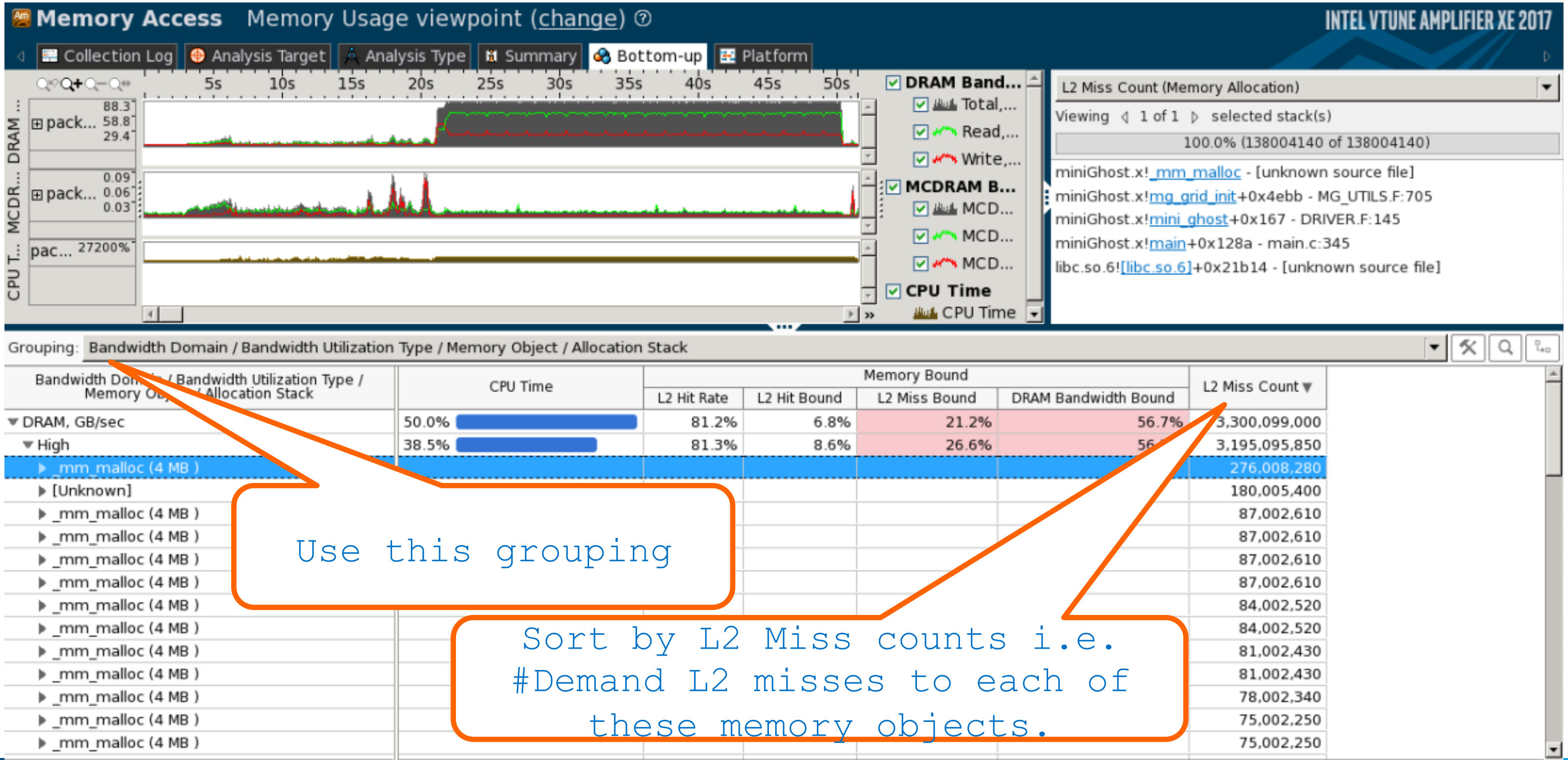
Memory Bandwidth on KNL

High Bandwidth Memory

- Want to maximize utilization
- Find high use memory objects using Intel® VTune™ Amplifier
- Allocate high use memory objects into HBM
  - Memkind library http://memkind.github.io/memkind
    - Also includes AutoHBW
  - Use numactl

Identifying high bandwidth memory objects (1/3)

Memory object analysis: DDR only


Memory Access   Memory Usage viewpoint (change)
INTEL VTUNE AMPLIFIER XE 2017
Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Platform
5s 10s 15s 20s 25s 30s 35s 40s 45s 50s
DRAM ... pack... 88.3 58.8 29.4
MCDR... pack... 0.09 0.06 0.03
CPU T... pac... 27200%
DRAM Band... Total,... Read,... Write,...
MCDRAM B... MCD... MCD... MCD...
CPU Time CPU Time

L2 Miss Count (Memory Allocation)

Viewing 1 of 1 selected stack(s)

100.0% (138004140 of 138004140)

```
miniGhost.x!_mm_malloc - [unknown source file]
miniGhost.x!mg_grid_init+0x4ebb - MG_UTILS.F:705
miniGhost.x!mini_ghost+0x167 - DRIVER.F:145
miniGhost.x!main+0x128a - main.c:345
libc.so.6![libc.so.6]+0x21b14 - [unknown source file]
```

Grouping: Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack

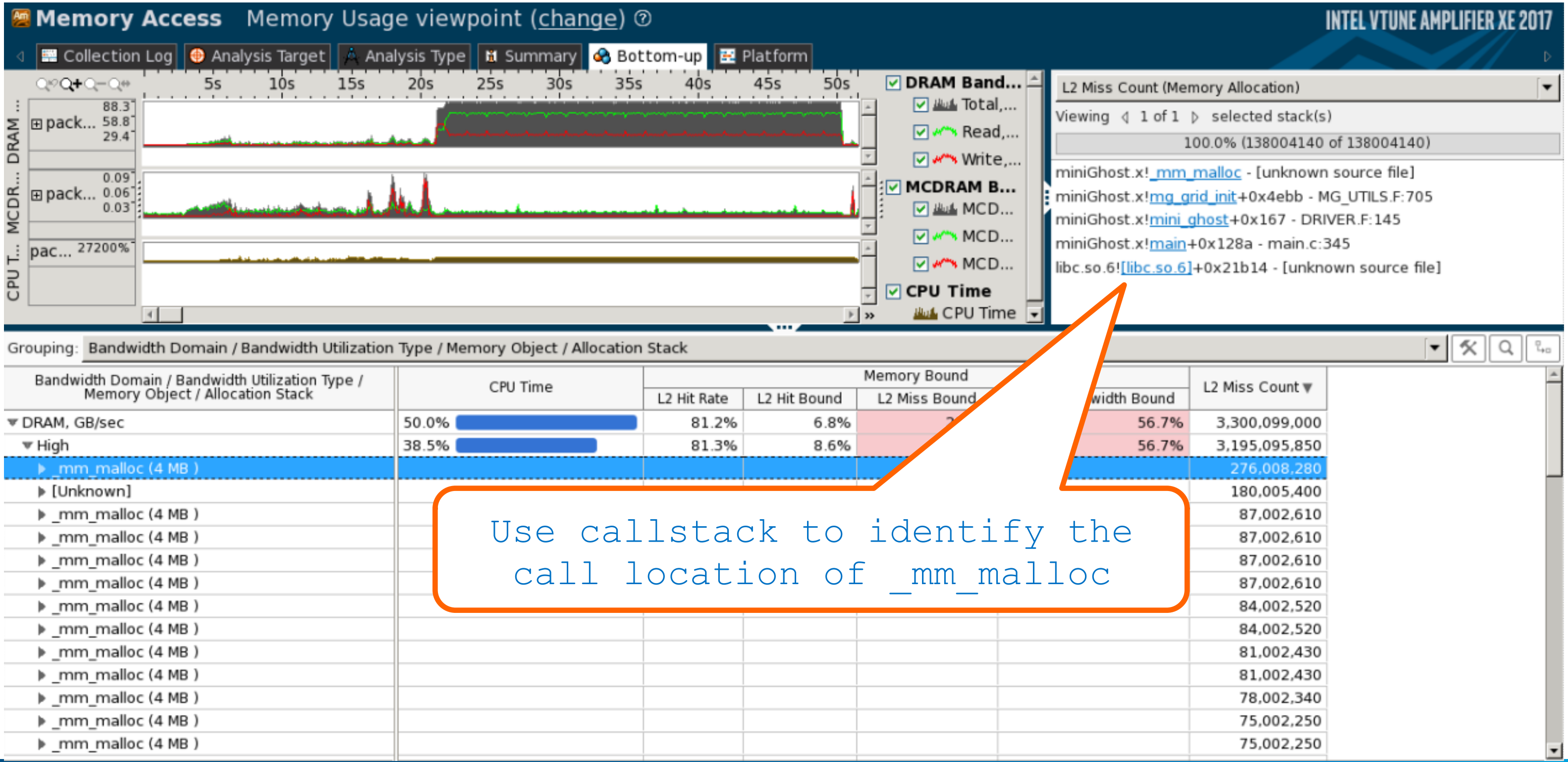
Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack	CPU Time	Memory Bound				L2 Miss Count ▼
		L2 Hit Rate	L2 Hit Bound	L2 Miss Bound	DRAM Bandwidth Bound	
▼ DRAM, GB/sec	50.0%	81.2%	6.8%	21.2%	56.7%	3,300,099,000
▼ High	38.5%	81.3%	8.6%	26.6%	56	3,195,095,850
▶ _mm_malloc (4 MB)						276,008,280
▶ [Unknown]						180,005,400
▶ _mm_malloc (4 MB)						87,002,610
▶ _mm_malloc (4 MB)						87,002,610
▶ _mm_malloc (4 MB)						87,002,610
▶ _mm_malloc (4 MB)						87,002,610
▶ _mm_malloc (4 MB)						84,002,520
▶ _mm_malloc (4 MB)						84,002,520
▶ _mm_malloc (4 MB)						81,002,430
▶ _mm_malloc (4 MB)						81,002,430
▶ _mm_malloc (4 MB)						78,002,340
▶ _mm_malloc (4 MB)						75,002,250
▶ _mm_malloc (4 MB)						75,002,250

Use this grouping

Sort by L2 Miss counts i.e. #Demand L2 misses to each of these memory objects.

Identifying high bandwidth memory objects (2/3)

Memory object analysis: DDR only



L2 Miss Count (Memory Allocation)

Viewing 1 of 1 selected stack(s)

100.0% (138004140 of 138004140)

miniGhost.x!_mm_malloc - [unknown source file]
miniGhost.x!mg_grid_init+0x4ebb - MG_UTILS.F:705
miniGhost.x!mini_ghost+0x167 - DRIVER.F:145
miniGhost.x!main+0x128a - main.c:345
libc.so.6![libc.so.6]+0x21b14 - [unknown source file]

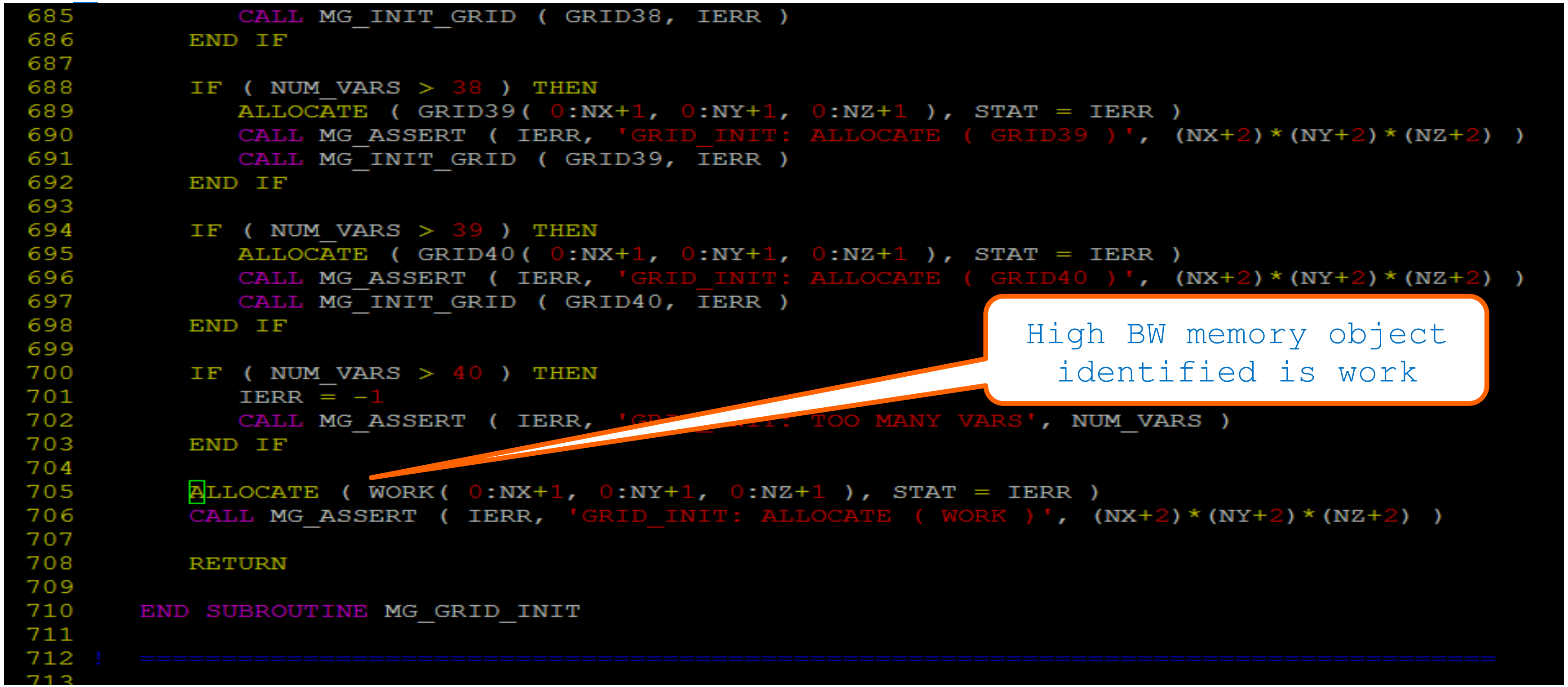
Grouping: Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack

Bandwidth Domain / Bandwidth Utilization Type / Memory Object / Allocation Stack	CPU Time	Memory Bound					L2 Miss Count
		L2 Hit Rate	L2 Hit Bound	L2 Miss Bound		width Bound	
DRAM, GB/sec	50.0%	81.2%	6.8%	2		56.7%	3,300,099,000
High	38.5%	81.3%	8.6%			56.7%	3,195,095,850
_mm_malloc (4 MB)							276,008,280
[Unknown]							180,005,400
_mm_malloc (4 MB)							87,002,610
_mm_malloc (4 MB)							87,002,610
_mm_malloc (4 MB)							87,002,610
_mm_malloc (4 MB)							87,002,610
_mm_malloc (4 MB)							84,002,520
_mm_malloc (4 MB)							84,002,520
_mm_malloc (4 MB)							81,002,430
_mm_malloc (4 MB)							81,002,430
_mm_malloc (4 MB)							78,002,340
_mm_malloc (4 MB)							75,002,250
_mm_malloc (4 MB)							75,002,250

Use callstack to identify the call location of _mm_malloc

Identifying high bandwidth memory objects (3/3)

MG_UTILS.F



```
         CALL MG_INIT_GRID ( GRID38, IERR )
      END IF

      IF ( NUM_VARS > 38 ) THEN
         ALLOCATE ( GRID39( 0:NX+1, 0:NY+1, 0:NZ+1 ), STAT = IERR )
         CALL MG_ASSERT ( IERR, 'GRID_INIT: ALLOCATE ( GRID39 )', (NX+2)*(NY+2)*(NZ+2) )
         CALL MG_INIT_GRID ( GRID39, IERR )
      END IF

      IF ( NUM_VARS > 39 ) THEN
         ALLOCATE ( GRID40( 0:NX+1, 0:NY+1, 0:NZ+1 ), STAT = IERR )
         CALL MG_ASSERT ( IERR, 'GRID_INIT: ALLOCATE ( GRID40 )', (NX+2)*(NY+2)*(NZ+2) )
         CALL MG_INIT_GRID ( GRID40, IERR )
      END IF

      IF ( NUM_VARS > 40 ) THEN
         IERR = -1
         CALL MG_ASSERT ( IERR, 'GRID_INIT: TOO MANY VARS', NUM_VARS )
      END IF

      ALLOCATE ( WORK( 0:NX+1, 0:NY+1, 0:NZ+1 ), STAT = IERR )
      CALL MG_ASSERT ( IERR, 'GRID_INIT: ALLOCATE ( WORK )', (NX+2)*(NY+2)*(NZ+2) )

      RETURN

   END SUBROUTINE MG_GRID_INIT

!  ==========================================================================
```

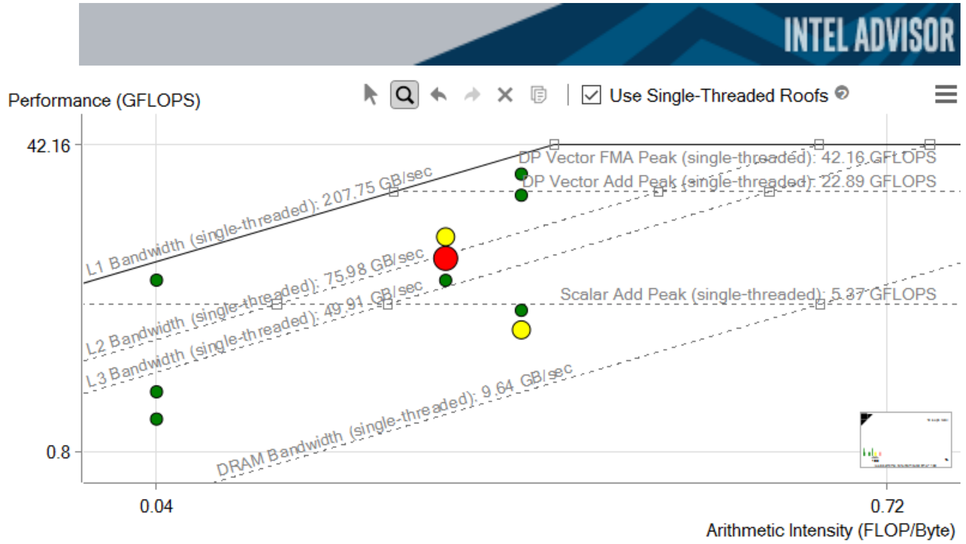
Roofline Analysis Using Intel® Advisor

Find Effective Optimization Strategies

Intel® Advisor: Cache-aware roofline analysis

Roofline Performance Insights

- Highlights poor performing loops
- Shows performance "headroom" for each loop
  - Which can be improved
  - Which are worth improving
- Shows likely causes of bottlenecks
- Suggests next optimization steps



Find Effective Optimization Strategies

Intel® Advisor: Cache-aware roofline analysis

Roofs Show Platform Limits

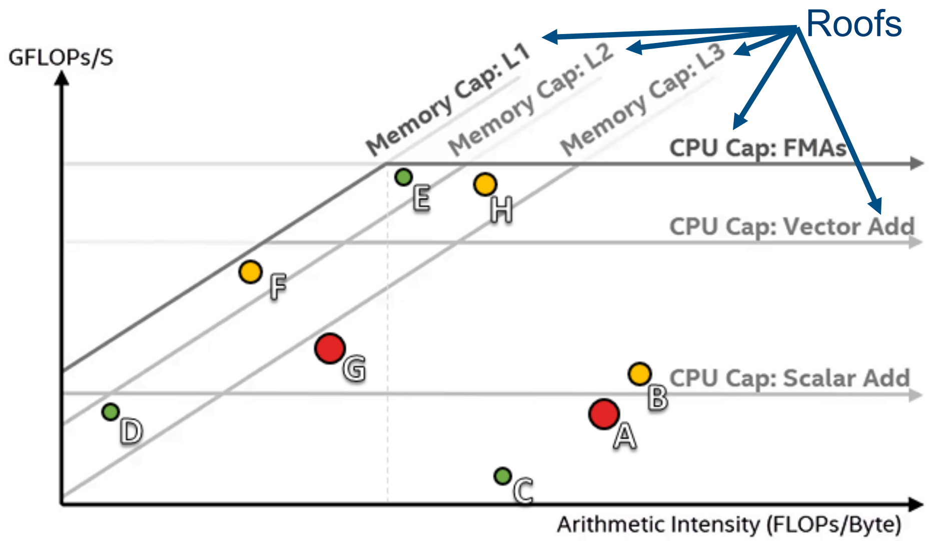
- Memory, cache & compute limits

Dots Are Loops

- Bigger, red dots take more time so optimization has a bigger impact
- Dots farther from a roof have more room for improvement

Higher Dot = Higher GFLOPs/sec

- Optimization moves dots up
- Algorithmic changes move dots horizontally



Which loops should we optimize?

- A and G are the best candidates
- B has room to improve, but will have less impact
- E, C, D, and H are poor candidates

Roofline tutorial video

Create Faster HPC, Cloud, and AI Software

What's New in Intel® Parallel Studio XE 2018 Beta

Get More Performance from New Hardware

- Use fast **AVX-512** instructions on **Intel® Xeon®** and **Xeon Phi™** processors
- Accelerate MPI applications with **Intel® Omni-Path** Architecture support

Discover Untapped Performance Faster

- **Intel® Advisor –** Use Roofline analysis to find high impact, but under optimized loops
- **Application Snapshot –** Get quick answers: Does my hybrid code need optimization?
- **Intel® VTune™ Amplifier** – Profile private clouds with Docker* containers, Java* daemons

Boost Machine Learning Application Performance

- **Intel® Data Analytics Acceleration Library** – Speed machine learning with new optimized algorithms
- **Intel® Distribution for Python* -** Accelerate Python code using fast NumPy/SciPy and scikit-learn packages

Latest Standards and IDEs

- **C++2017** draft parallelizes and vectorizes C++ easily using **Parallel STL***
- Full **Fortran* 2008**, **Fortran 2015** draft
- **OpenMP* 5.0** draft, **Microsoft Visual Studio* 2017**

And much more*…

Register for Beta at: http://intel.ly/intel-parallel-studio-xe-2018-beta

Optimization Notice

* See Release Notes for the full list with further updates and new features.

Legal Disclaimer & Optimization Notice

intel®

experience what's inside™