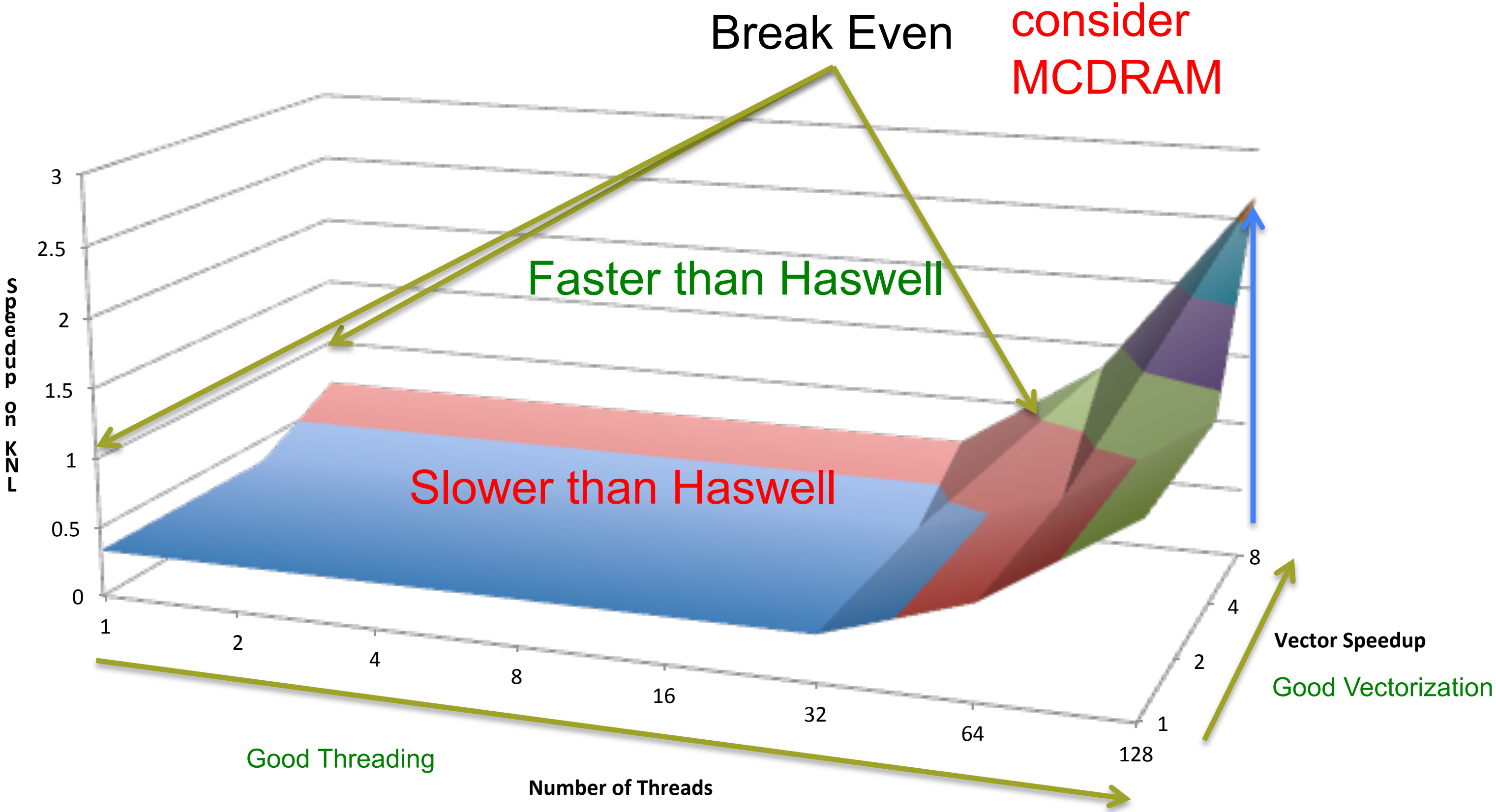
What You Need to Know About KNL

John M. Levesque
Director
Cray Supercomputing Center of Excellence
Trinity
Member of CTO Office

Setting expectations, when will KNL out-preform state-of-the-art Xeon

- Xeon is faster
  - Haswell 2.3 GHZ
  - KNL 1.4 GHZ
- KNL has more cores
  - Haswell has 32 cores (2-16 core sockets)
  - KNL has 68 cores
- KNL has more threads
  - Haswell allows 64 threads – 32x2 Hyperthreads
  - KNL allows 272 threads – 68x4 Hyperthreads
- KNL has longer SIMD unit
  - Haswell has SIMD unit of length 4
  - KNL has SIMD unit of length 8
- KNL has High Bandwidth Memory – 4-5 times Xeon bandwidth
- KNL has less low level cache

KNL versus Haswell
Simple Amdahl's Law



This does not consider MCDRAM

Important Architectural Data for HPC chips

Arch	MIMD Procs	SIMD DP Width	Close Memory (Cache) KB/ core	HBM GB	HBM GB/ sec	Main Memory GB	Main Memory GB/sec
K20X	14	32	90	6	250	-	
K40	15	32	96	12	288	-	
K80(/2)	15	32	96	12	240	-	
P100	56	32	73	16	720	-	
Ivybridge	8	2	2887	-	-	128	100
Haswell	16	4	2887	-	-	256	120
KNL (MCDRAM)	68	8	557 (25264MB)	16	480 (329)	384	80

Important Data for Haswell and KNL

	Latency Haswell Nanosec (clocks)	Haswell Size /core	Latency KNL Nanosec (clocks)	KNL Size /core	Bandwidth Haswell GB/sec Stream Triad	Bandwidth KNL GB/sec Stream Triad
L1 Cache	2.7 (6)	32KB	2.9 (4)	32KB		
L2 Cache	8.11 (19)	256KB	13.6 (19)	512KB		
MCDRAM (Cache)	24.35 (56) (Level 3)	2.5MB	173.5 (243)	242MB		329
MCDRAM (Flat)			174.2 (244)	242MB		486
DDR (Flat)			151.3 (212)		102	90
DDR (Cache)						59

Most Important Feature of KNL is MCDRAM

- **Most of our applications are memory bandwidth limited.**
- **MCDRAM presents a smaller high bandwidth memory that can be used to address memory bandwidth issues**
  - When used as a separate address space (FLAT) it has the highest Triad Stream transfer rate of 480 GB/sec and a slightly higher latency of 174 nanoseconds than DDR memory
  - When used as cache (CACHE) it has a lower Triad Stream transfer rate of 329 GB/sec and about the same latency MCDRAM Flat

MCDRAM as Cache

- Latency is much longer than typical caches
- Bandwidth is excellent albeit slower than MCDRAM as separate address space (480 versus 325).
- Only way to transfer pages to MCDRAM without going through low level caches is to use it as Cache
- MCDRAM as Cache has one major potential hazard. It is direct mapped and some applications may experience significant degradation in performance. This degradation is caused by some nodes of a large parallel run experiencing cache conflicts and introducing load imbalance.

MCDRAM as Flat

- **Latency is approximately the same as DDR latency – little slower**
- **Bandwidth is excellent – five times faster than DDR.**
- **Two considerations**
  - When application and data require less than 16 GBytes
    - Easy to use - numctl --membind = 1
  - When application and data require more than 16 GBytes
    - Very difficult to get to run better than using MCDRAM as Cache
    - All Flat means that low level cache is as poor as accelerators
    - 50% or 75% Flat should work better; however, identifying which arrays to put into Flat is a complex problem
    - Using Flat as a scratch pad, swapping data in and out ruins contents of low level – Still being considered by some application developers
- **Using MCDRAM as flat can be used to increase the total memory on the node – when memory size is more important than maximum performance.**

What you need to know about MCDRAM

- **Very few applications (if any) have shown that using MCDRAM as Flat gives an improvement over using MCDRAM as Cache**
- **If your application plus data uses less than 16 Gbytes, use Flat with job launch command numactl --membind=1**
- **If your application plus data uses more than 16 Gbytes, use Cache**
- **If you use Cache – check to see if your application is susceptible to issues caused by Direct Mapped Cache**
  - Run application with DDR (numactl --membind=0) only up to 100 to 1000 nodes
  - Run application with Cache up to 100 to 1000 nodes
  - If DDR only scales better than Cache then Cache has issues with cache conflicts

Closer Look at MCDRAM

Presentation by Nathan Wichmann

Different Triads

- **Triad**
  - a(j) = b(j) + scalar*c(j)
  - Simple STREAM Triad with 3 stride 1 arrays
- **Triad – Stride -1 Gather**
  - a(j) = b(j) + scalar*c(index1(j))
  - Triad with one arrays loaded using index array, but index is still a stride-1 pattern
- **Triad – Random Gather**
  - a(j) = b(j) + scalar*c(indexr(j))
  - Triad with one arrays loaded using index array, but index is still a **random** pattern
- **Triad – Full Random Gather / Scatter**
  - a(indexr(j)) = b(indexr(j)) + scalar*c(indexr(j))
  - Full random pattern on all three arrays

Array Sizes and footprints

- **Each of the arrays are 2M elements (15.26 Mbytes)**
  - Simple Triad uses 3 arrays and thus touches a total of ~46 Mbytes
  - Triads that use index use 4 arrays and thus a total of ~61 Mbytes
- **Each core works on its own copy for a total footprint of 3.1-4.1 Gbytes per node**
  - No L3 cache reuse on Haswell
  - But should be able to easily reside inside of 50% of MCDRAM, no matter the mode
- **Ideal problem to showcase the advantage of MCDRAM**
  - High bandwidth test that fits in MCDRAM but cannot be blocked for L3 on XEON

Different KNL modes

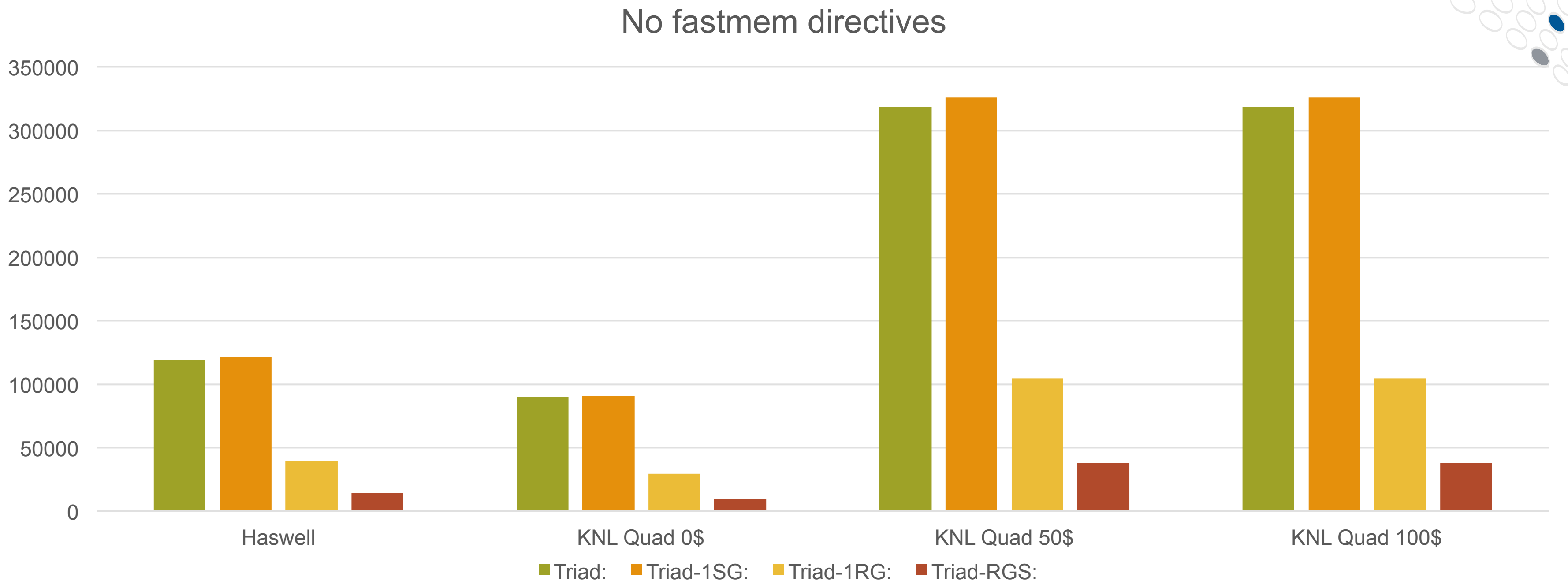
- **Quad – 0% cache**
  - Also known as “flat mode”
  - All of MCDRAM must be explicitly targeted
- **Quad – 100% cache**
  - All of MCDRAM is a direct mapped cache
  - MCDRAM is not available for capacity
  - Don’t have to do anything to the codes
- **Quad – 50% cache**
  - Also known as “hybrid mode”
  - 50% of MCDRAM is a direct mapped cache
  - 50% of MCDRAM must be explicitly targeted
  - Pragmatic compromise?

Fastmem directives

- Add the !DIR$ ATTRIBUTES FASTMEM :: X directive when you declare the arrays.

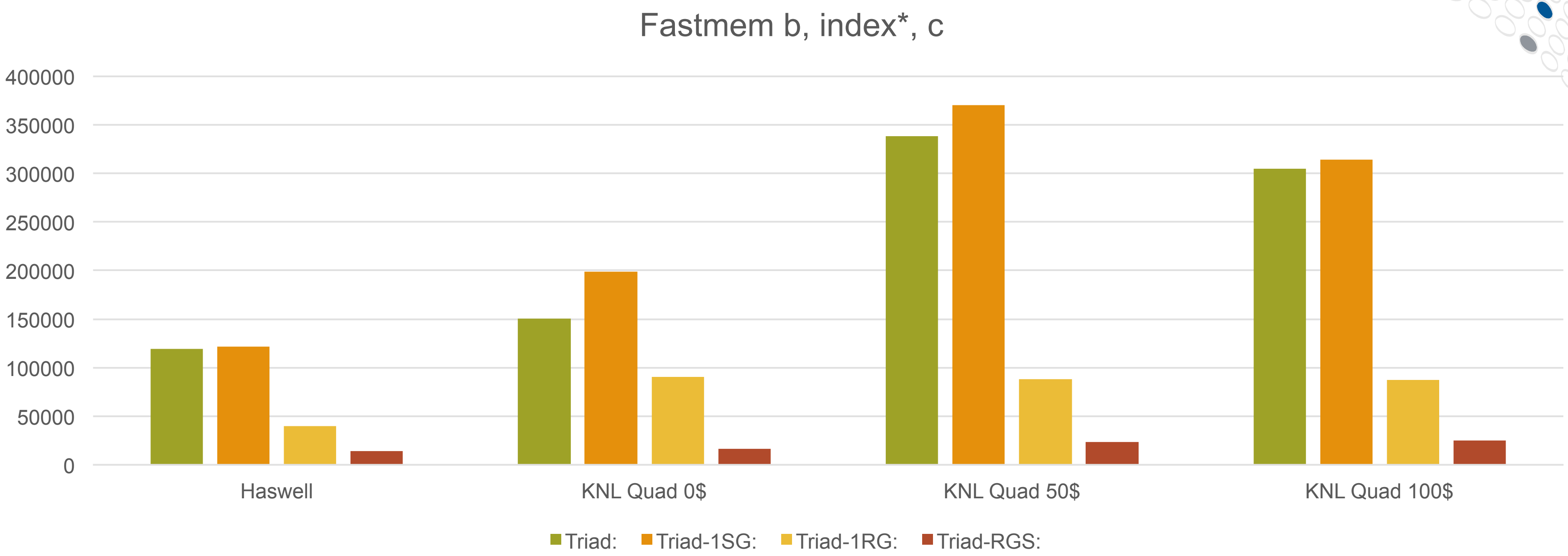
```
module swap PrgEnv-cray PrgEnv-intel
#Have to find a memkind library until it gets added to CLE (which will be soon).
module use /cray/css/users/kjt/opt/modulefiles
module load memkind

module mods
    real(kind=8), allocatable :: a(:),b(:),c(:),d(:),x(:)
!dir$ attributes fastmem :: a,b,c,d,x
  end module mods
```

No arrays in explicit fastmem



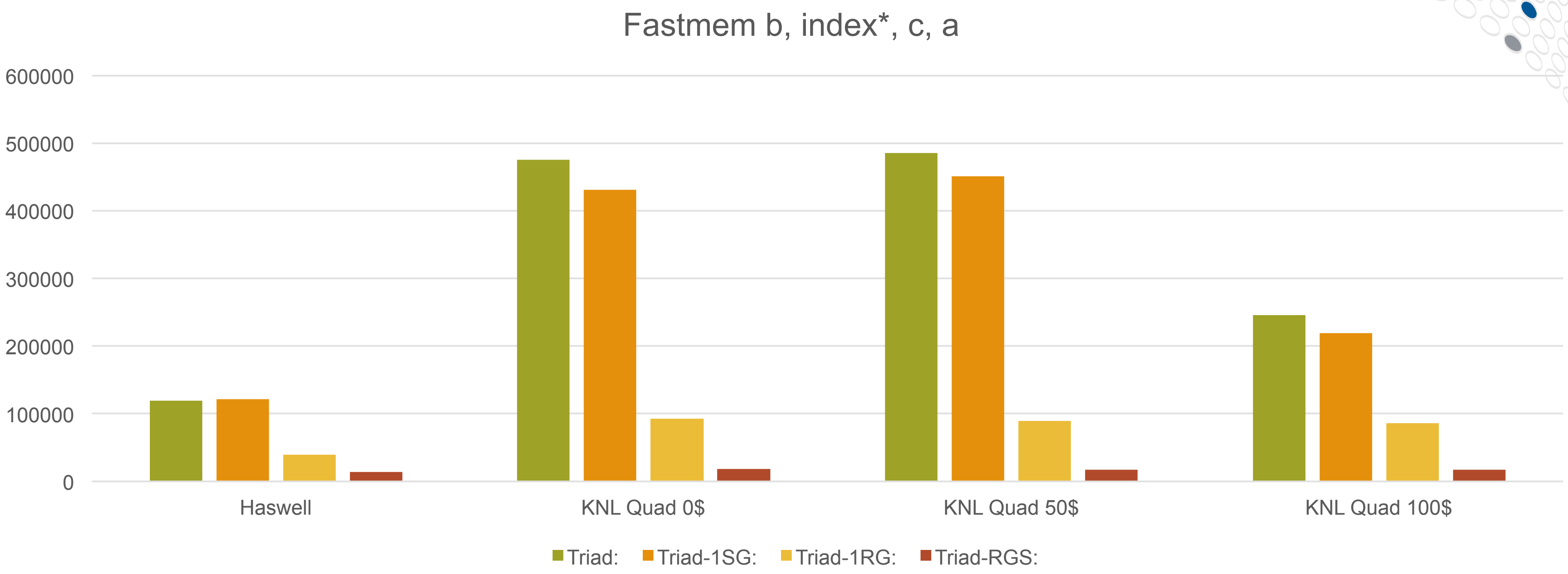
- Haswell is faster than 0% cache
- But 50% and 100% cache modes work well to capture available reuse

Most arrays in explicit fastmem



- **Even with most of the arrays in fastmem, 0% still falls short of cache for stride-1 cases**

ALL arrays in explicit fastmem



- **0% mode finally does very well with all arrays in fastmem**
  - But 50% cache easily keeps up because its fastmem can also hold the data
- **100% cache does poorly if combined with fastmem directive**
  - Why?

Let’s prematurely jump to some observations and conclusions

0% cache mode

- **Maximizes amount of explicit space**
- **Highest performance if you can fit *all* of your data**
- **But performance can quickly suffer if you miss some MCDRAM reuse opportunity**
  - Amdahl’s Law

- **Conclusion**
  - If you full data set can fit inside of 16 Gbytes per node, just use this mode

100% cache mode

- **Maximizes amount of cache space**
- **Does not require any directives**
- **It seem to capture reuse**

- **Conclusion**
  - A great starting point!
    - You might not be able to do any better than this

50% cache mode

- **Cuts cache space by half**
  - How often will that matter?
- **Captures a lot of reuse with no work**
- **Allows for some incremental upside**
- **0% cache will beat this only if 100% of bandwidth sensitive data fits in 16 GB but does not fit into 8GB**

- **Conclusion**
  - Gives good performance in a number of situations, but allows for one to experiment with fastmem directives
  - The mode has not been adequately tested yet

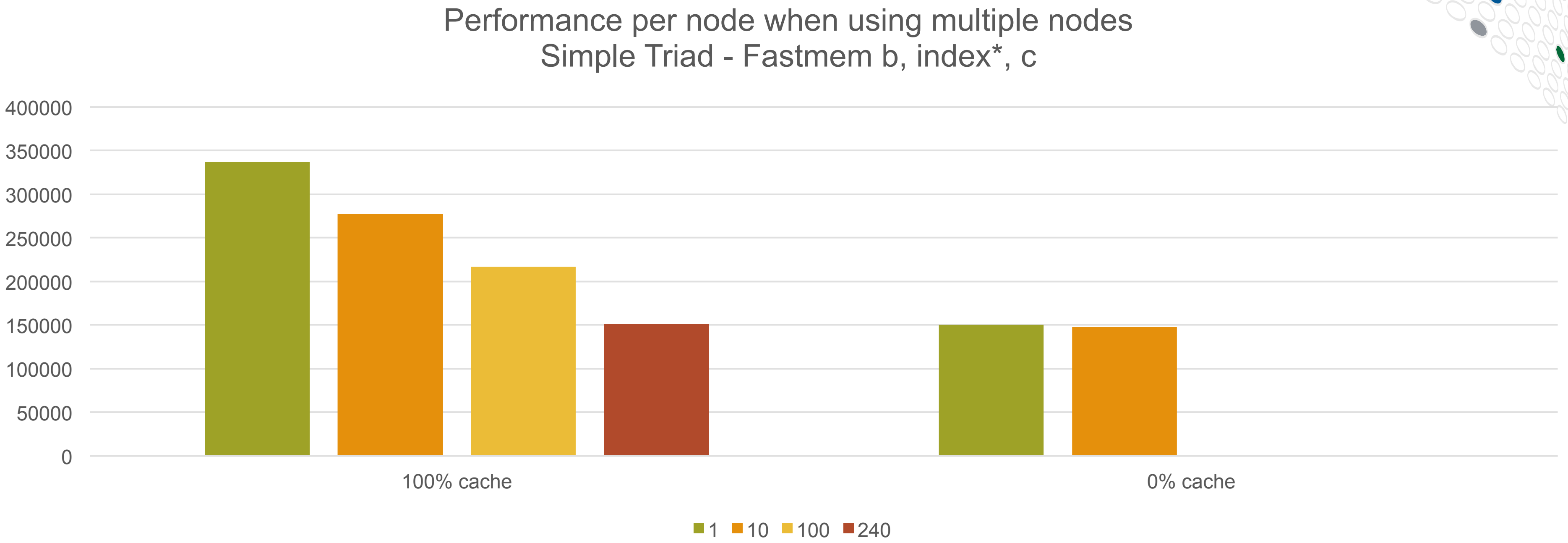
That sounds good, but wait… just one problem

Single node vs multi-node

- All of these tests were run on just a single node

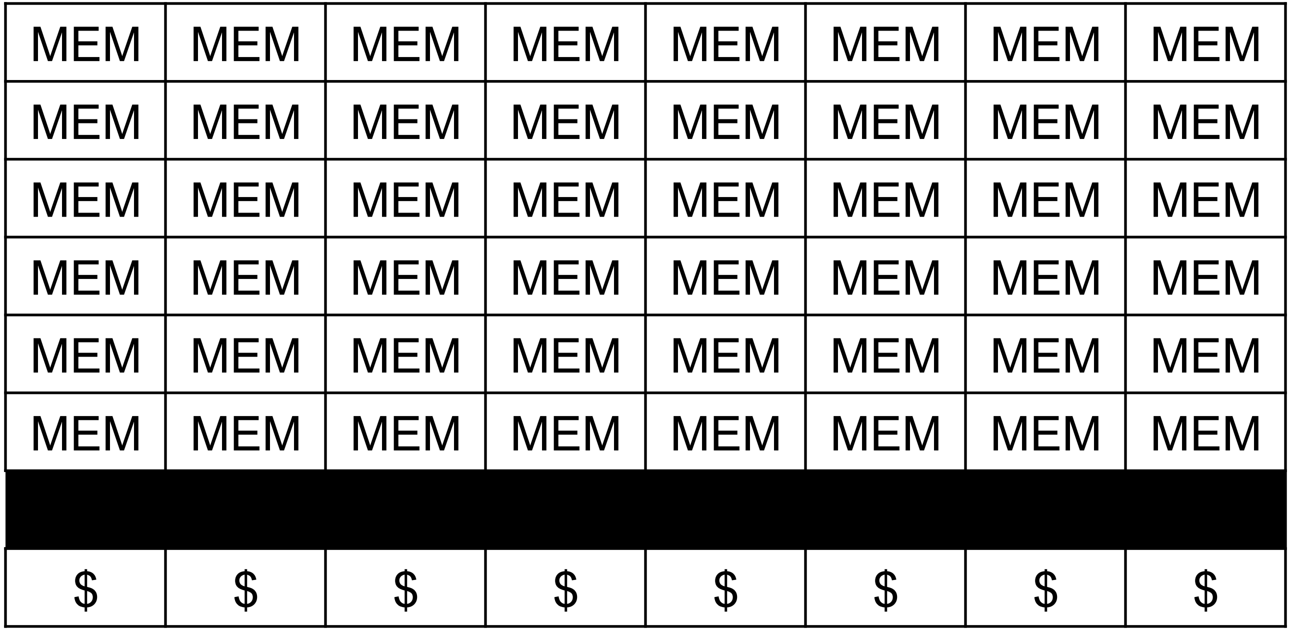
- Does the picture change if we run on more than one node?

Increase fastmem for 100% cache mode



- The performance of cache drops significantly when more nodes are used!
- Why?

Basic picture of memory and cache



MEM	MEM	MEM	MEM	MEM	MEM	MEM	MEM
MEM	MEM	MEM	MEM	MEM	MEM	MEM	MEM
MEM	MEM	MEM	MEM	MEM	MEM	MEM	MEM
MEM	MEM	MEM	MEM	MEM	MEM	MEM	MEM
MEM	MEM	MEM	MEM	MEM	MEM	MEM	MEM
MEM	MEM	MEM	MEM	MEM	MEM	MEM	MEM
$	$	$	$	$	$	$	$

- **There are 6 memory locations that map to the same location in cache**
  - Assumes a 96 GB DRAM config. Larger memory have more
- **Cache is direct mapped. I.E. there is only 1 “way”**

Start filling pages in memory

					PE3's A		
						PE5's A	
	PE4's A						
			PE2's A				
PE1's A							
$PE1's A	$PE4's A	$	$PE2's A	$	$PE3's A	$PE5's A	$

- **OS starts to place pages that contain the variable "A" for each PE as those PEs reach the allocation statement**
  - Each page placement is more or less "random" based on when various Pes arrive at the allocate, and the order in which the pages were free, perhaps even by a previous program

Conflicts are bound to happen

			PE6’s A				
					PE3’s A		
						PE5’s A	
	PE4’s A						
			PE2’s A				
PE1’s A							

$PE1’s A	$PE4’s A	$	$ conflict	$	$PE3’s A	$PE5’s A	$

- If two PEs A are aliased to the same cache location, then thrashing may occur if those PEs are both reuse A at “about” the same time

Cache trashing will occur, but how often?

- **Good news: The cache is really large, so there is a relatively low probability of an aliasing conflict occurring**
  - Conflict probability is also a function of size of the data being reused
  - This is why we often don't observe this on single node runs

- **Bad news: All of these are NON-ZERO probabilities, and the dice are rolled on every node in the job**
  - As the number of nodes in the job increases, the probability and aliasing problems will occur approaches 100%
    - If one node sees it in one part of the code, all nodes will see it via synchronization
    - Furthermore the slow PEs can be different for different parts of the code
  - Hyperthreading only makes the problem worse as more threads are putting more pressure on the cache levels
  - OpenMP may lower the probability in some cases, but since different threads will hit different memory sections, the problem will still occur
  - This is not limited to "high-bandwidth" arrays, but even those arrays that may get good L2 cache hit rates

Direct mapped cache causing scaling problems

- **If your performance is impacted by the effectiveness of the MCDRAM cache, you may experience scaling problems**
  - This will likely show up an communication, but will be because of synchronization, not bandwidth or latency constraints
- **Normal profiling may not point to the offending compute region**
  - Only a few PEs might be slow, and thus that signal could be drowned out by the other Pes in the job
- **We may be able to limit this problem by running in 50% cache and putting some arrays into fastmem**
  - But this remains a theory and so far has not worked out in at least one case

Conclusions

- **Flat mode is good if your entire data set can fit into 16GB**
- **Flat mode seems likely to be unforgiving if some bandwidth data does not fit**
- **Cache mode seems to capture reuse well**
  - And it requires no work on the part of the benchmarker
- **But is highly susceptible to thrashing if important data aliases to the same location**
  - This becomes more and more likely as node counts increase
- **Each mode has serious issues and seem likely to cause performance problems as one tries to use more nodes or run larger problems**

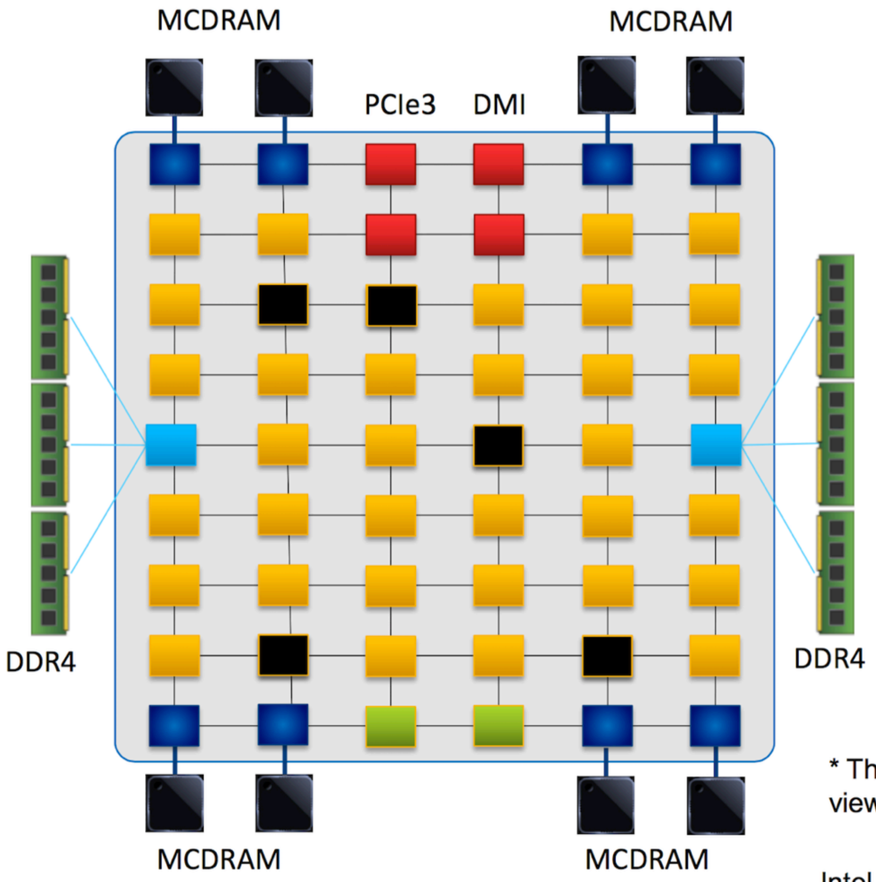
Next most important feature of KNL

- **Vectorization**
  - KNL can generate up to 16 results each clock cycle
    - Eight Adds and Eight Multiplies
  - KNL has predicated execution
    - Has a mask register to handle IF conditions
  - KNL has some hardware assist for gather/scatter
    - Still vectorizing loops with indirect addressing does not give good performance
  - Vector operations on operands aligned on cache boundaries perform better than misaligned operands.
- **Vectorization will not perform as expected if the loop is memory bandwidth limited**
  - Cache utilization is more important than vectorization

Next most important feature of KNL

- **Lots of hardware threads on the node**
  - The KNL node is a 2-D mesh of 36 tiles
    - Each tile has two cores
      - Each cores allows four hardware threads
    - On current systems two of the tiles are disabled, resulting in 34 usable tiles
  - Each tile has
    - Two cores with Level 1 cache
    - Shared Level 2 cache
    - Cache Homing Agent (CHA) which handles cache coherency across the node

Tile To NUMA Node Mapping Concepts



- KNL has 38 processing tiles per chip
- 68 core part will have 4* tiles disabled at the factory
- ApicID is hardwired into each tile
- Disabled tiles will vary from part to part
- As a result, ApicIDs will vary from part to part

🟨 Processing Tile

🟦 MCDram Controller

🟦 DDR4 Controller

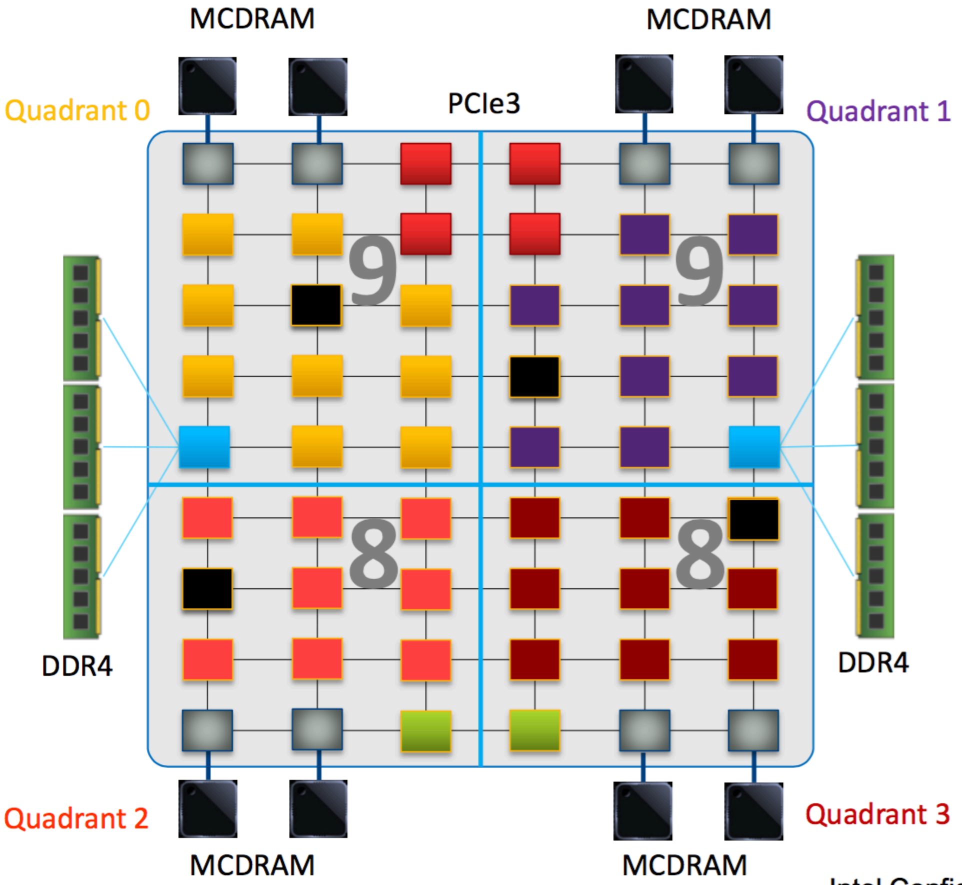
🟥 PCIe, DMI Interfaces

🟩 Other

⬛ Factory disabled

* This slide has animation showing the random location of 4 disabled tiles. When viewed in .pdf format it looks like 5 tiles have been disabled.

68 Core SNC4 Tile Mapping Example – Intel BIOS



- Like SNC2, it is up to the BIOS to define and implement the tile assignments
- There are no hard wired circuits within KNL as to how tiles are mapped to NUMA nodes
- Best configuration is to have 9 tiles per quadrant on the top and 8 tiles on the bottom
- This is the preferred model that will be implemented by Intel BIOS

Options for using all the hardware threads

- **All - MPI**
  - Works surprising well; however, typically MPI does not run well on hyper-threads – maybe 64 MPI tasks on the node is best; however, I have seen 128 MPI tasks on the node run better than 64. Should be tried for those applications that have no shared memory threading before spending a lot of work on application.
- **All – Shared Memory threads**
  - VERY Few applications are threaded well enough to utilize the KNL node in this way (TBD)
- **Hybrid MPI/Shared Memory threads**
  - Best approach for those applications that already have shared memory threading in the application
  - Need to find "Sweet Spot"

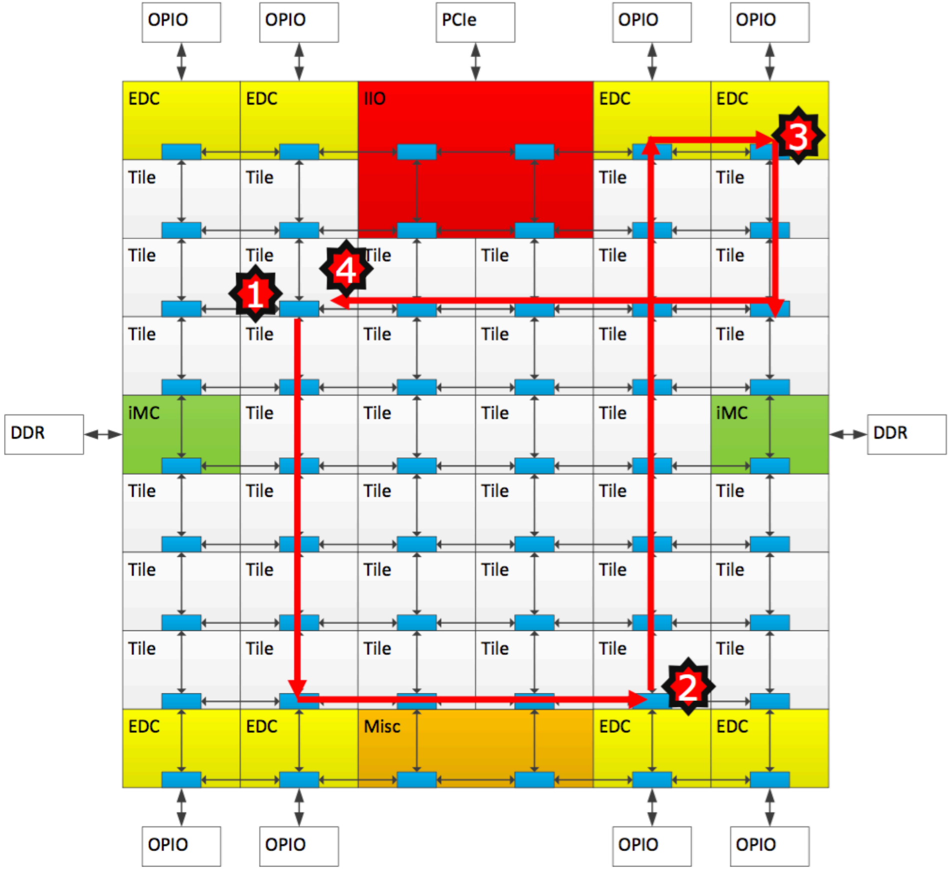
What you need to know about the KNL node

- **KNL has significant NUMA issues on the node which introduces inefficiencies in shared memory threading**
  - High level OpenMP with first touch are a necessity for good scaling
- **The tile is the closest thing to a UMA region – should we at least use 34 MPI tasks on the node with 2,4 or 8 threads per MPI task?**
- **A Quadrant is the next memory region when using SNC4**
  - NUMA issues within the Quadrant

Investigate Clustering Modes

- **Three Major cluster modes being used**
  - Quad
  - SNC4
  - SNC2
- **Couple more that are not used**
  - All to All
  - Hemisphere

Cluster mode: All-to-All



Address uniformly hashed across all distributed directories

Typical Read L2 miss

1. L2 miss encountered

2. Send request to the distributed directory

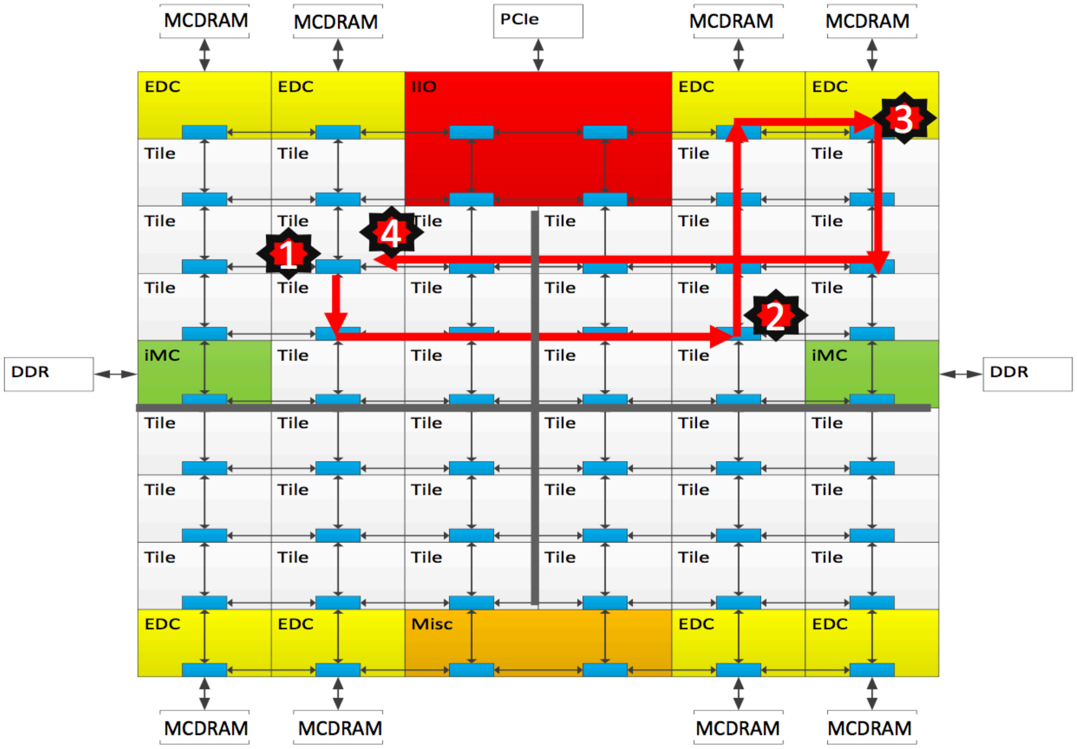
3. Miss in the directory. Forward to memory

4. Memory sends the data to initial requestor

Cluster – All to All

- Has lowest bandwidth and highest latency of all clustering modes
- Do not know of any application using this today

Cluster Mode: Quadrant



Chip divided into four virtual Quadrants

Address hashed to a Directory in the same quadrant as the Memory

Affinity between the Directory and Memory

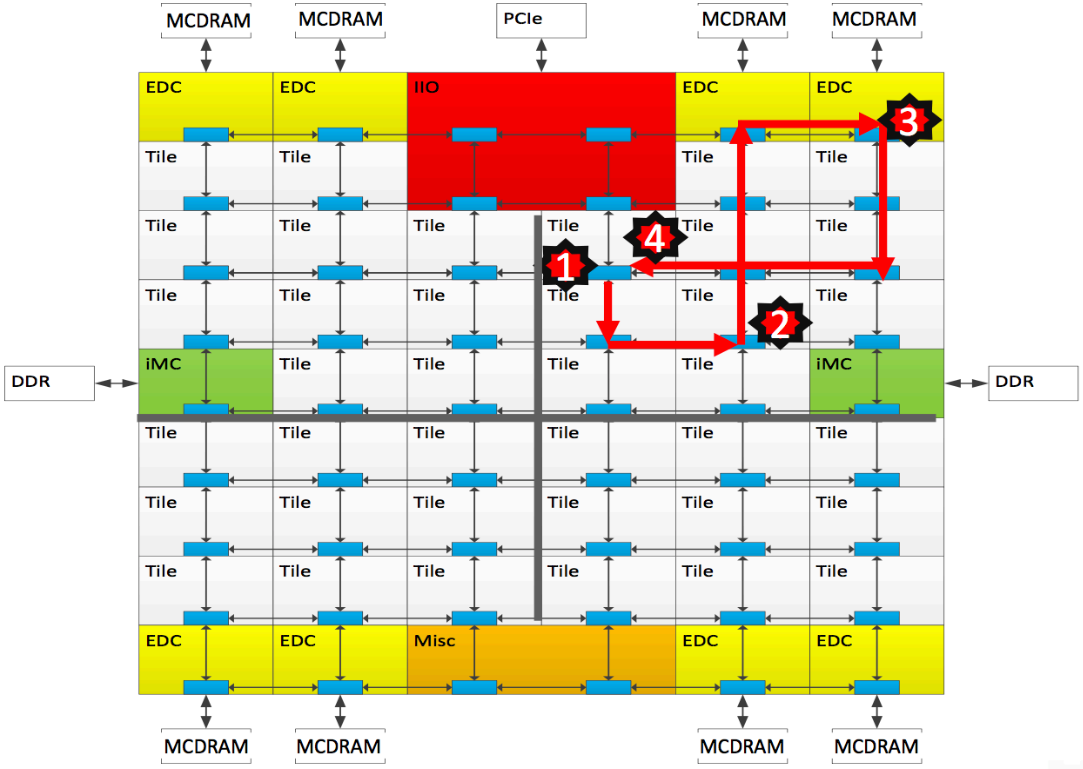
Lower latency and higher BW than all-to-all. SW Transparent.

1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return

Cluster – Quad

- **Allows a collection of cores within a quadrant to get all of the memory bandwidth**
  - Could be important for a bandwidth sensitive, load imbalanced mostly-MPI application.
    - Mostly MPI = greater than/or equal to 16 MPI tasks per node
- **Introduces additional NUMA affects when threading across more than one tile**

Cluster Mode: Sub-NUMA Clustering (SNC)



1) L2 miss, 2) Directory access, 3) Memory access, 4) Data return

Each Quadrant (Cluster) exposed as a separate NUMA domain to OS.

Looks analogous to 4-Socket Xeon

Affinity between Tile, Directory and Memory

Local communication. Lowest latency of all modes.

SW needs to NUMA optimize to get benefit.

Cluster – SNC2 – SNC4

- **Introduces 2 or 4 socket boundaries to node.**
  - Cons
    - Latency and Bandwidth between sockets reduced
    - Collection of cores can only access ¼ memory bandwidth and ¼ of DDR and ¼ MCDRAM efficiently
  - Pros
    - Assists in improving locality when threading across multiple tiles, when OMP_NUM_THREADS > 8
    - Has the best latency to memory of all the clustering modes

Simple Rules for Clustering

- **If the number of MPI tasks is greater than or equal to 32**
  - If your application is memory bandwidth limited, Quadrant would be best
  - If you application is latency bound, SNC4 should be best
- **If the number of MPI tasks is less than or equal to 16**
  - To reduce the NUMA effects SNC2 or SNC4 should be used
  - If your application is memory bandwidth limited, Quadrant would be best
- **Most of the time the differences in clustering modes will be less than 10% - willbe some outliers.**
- **Find out what best for you and stick with it**
- **Nightmare for operational system because re-provisioning takes a long long time (Today 20-25 minutes)**

A simple flowchart for examining an application with Perftools

Before We get started

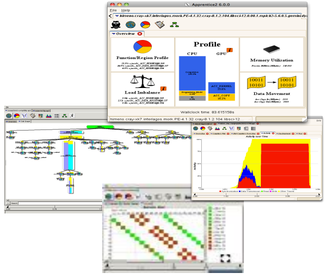
- **Many of the tools discussed will work with the Intel compiler; however, we highly recommend that you try to build with the Cray compiler, since the following only works with the Cray compiler**
  - Memory Analysis tools
  - Loop profiling
  - Reveal
  - Profiling of library groups

Cray Performance Tools

Heidi Poxon
Sr. Principal Engineer
Cray Programming Environment

Using Cray Performance Tools

Load modules to access software



Build and instrument program

Run program and view results

Two Modes of Use

- **CrayPat-lite** for novice users, or convenience

- **CrayPat** for in-depth performance investigation and tuning assistance

- Both offer:
  - Whole program analysis across many nodes
  - Indication of causes of problems
  - Suggestions of modifications for performance improvement

“Lite” Mode

Load performance tools instrumentation module

```
$ module load perftools-lite
```

Build program (no modification to makefile)

```
$ make
```

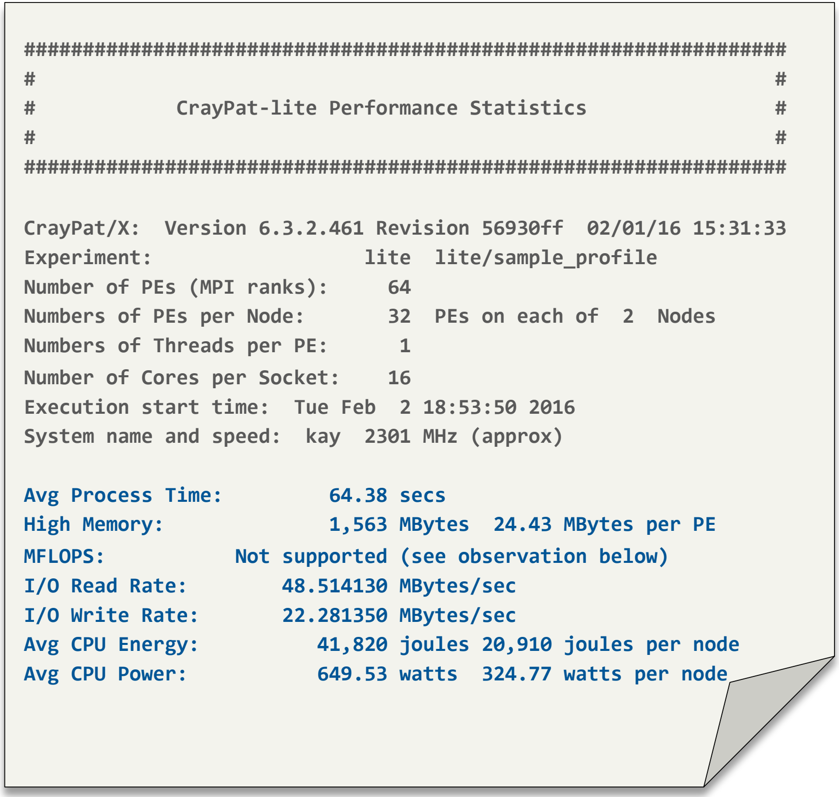
→ a.out (instrumented program)

Run program (no modification to batch script)

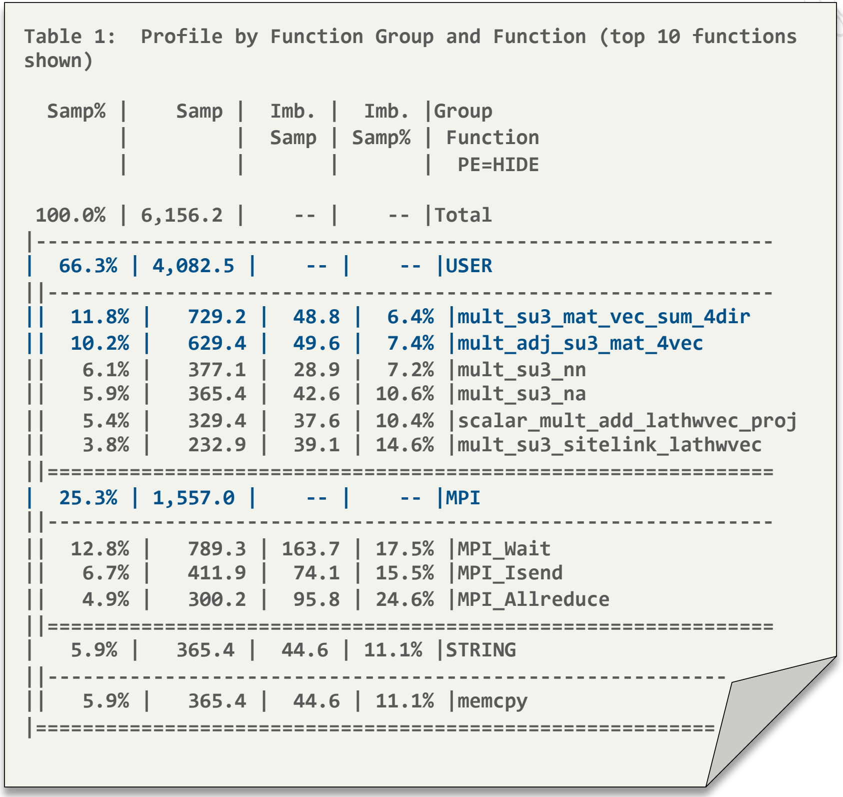
```
$ aprun a.out
```

→ Condensed report to stdout
a.out*.rpt (same as stdout)
a.out*.ap2
files

Example CrayPat-lite Output

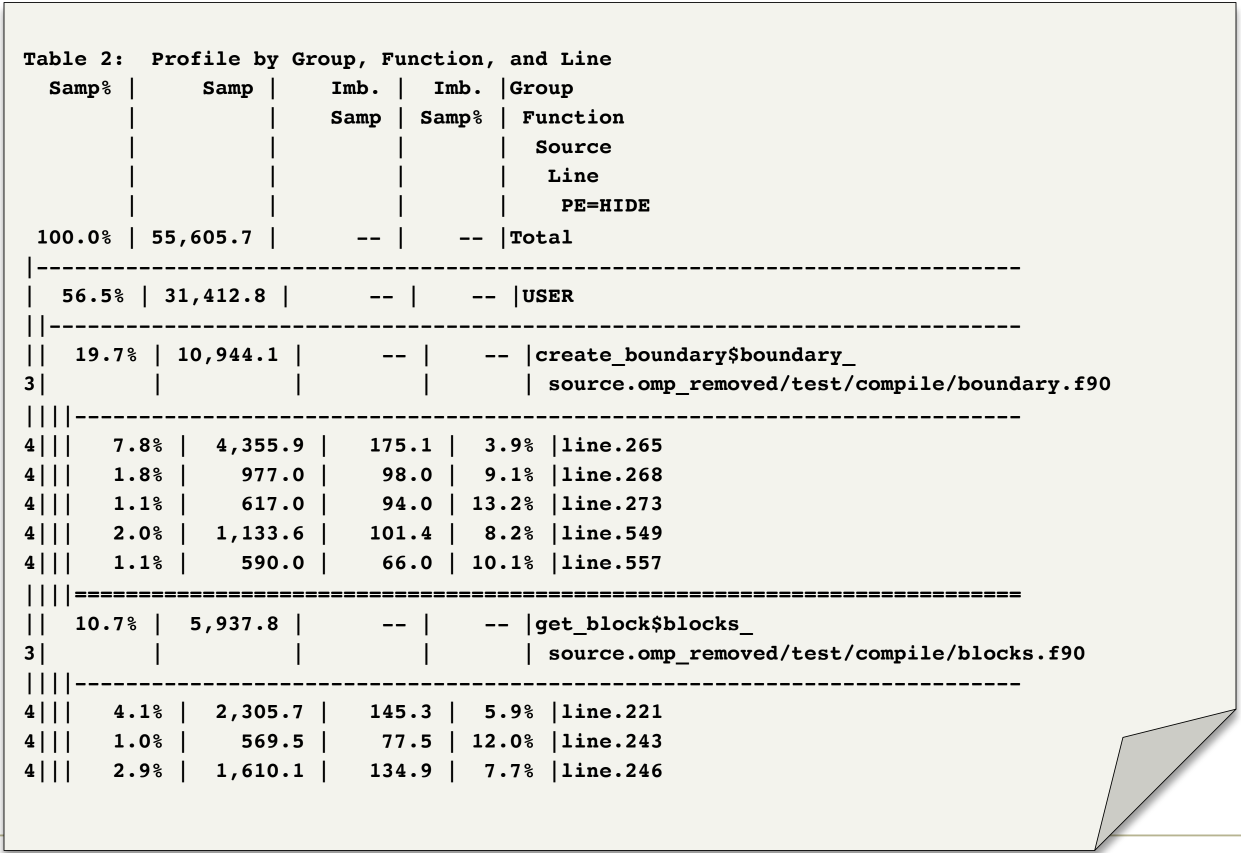


```
#################################################################
#                                                               #
#            CrayPat-lite Performance Statistics                #
#                                                               #
#################################################################

CrayPat/X:  Version 6.3.2.461 Revision 56930ff  02/01/16 15:31:33
Experiment:                  lite  lite/sample_profile
Number of PEs (MPI ranks):     64
Numbers of PEs per Node:       32  PEs on each of  2  Nodes
Numbers of Threads per PE:      1
Number of Cores per Socket:    16
Execution start time:  Tue Feb  2 18:53:50 2016
System name and speed:  kay  2301 MHz (approx)

Avg Process Time:          64.38 secs
High Memory:               1,563 MBytes  24.43 MBytes per PE
MFLOPS:             Not supported (see observation below)
I/O Read Rate:          48.514130 MBytes/sec
I/O Write Rate:         22.281350 MBytes/sec
Avg CPU Energy:           41,820 joules 20,910 joules per node
Avg CPU Power:            649.53 watts  324.77 watts per node
```



```
Table 1:  Profile by Function Group and Function (top 10 functions
shown)

  Samp% |     Samp |  Imb. |  Imb. |Group
        |          |  Samp | Samp% | Function
        |          |       |       |  PE=HIDE

 100.0% | 6,156.2 |    -- |    -- |Total
|-----------------------------------------------------------------
|  66.3% | 4,082.5 |    -- |    -- |USER
||----------------------------------------------------------------
||  11.8% |   729.2 |  48.8 |  6.4% |mult_su3_mat_vec_sum_4dir
||  10.2% |   629.4 |  49.6 |  7.4% |mult_adj_su3_mat_4vec
||   6.1% |   377.1 |  28.9 |  7.2% |mult_su3_nn
||   5.9% |   365.4 |  42.6 | 10.6% |mult_su3_na
||   5.4% |   329.4 |  37.6 | 10.4% |scalar_mult_add_lathwvec_proj
||   3.8% |   232.9 |  39.1 | 14.6% |mult_su3_sitelink_lathwvec
||================================================================
|  25.3% | 1,557.0 |    -- |    -- |MPI
||----------------------------------------------------------------
||  12.8% |   789.3 | 163.7 | 17.5% |MPI_Wait
||   6.7% |   411.9 |  74.1 | 15.5% |MPI_Isend
||   4.9% |   300.2 |  95.8 | 24.6% |MPI_Allreduce
||================================================================
|   5.9% |   365.4 |  44.6 | 11.1% |STRING
||----------------------------------------------------------
||   5.9% |   365.4 |  44.6 | 11.1% |memcpy
|==========================================================
```

Identify High Time Consuming Areas

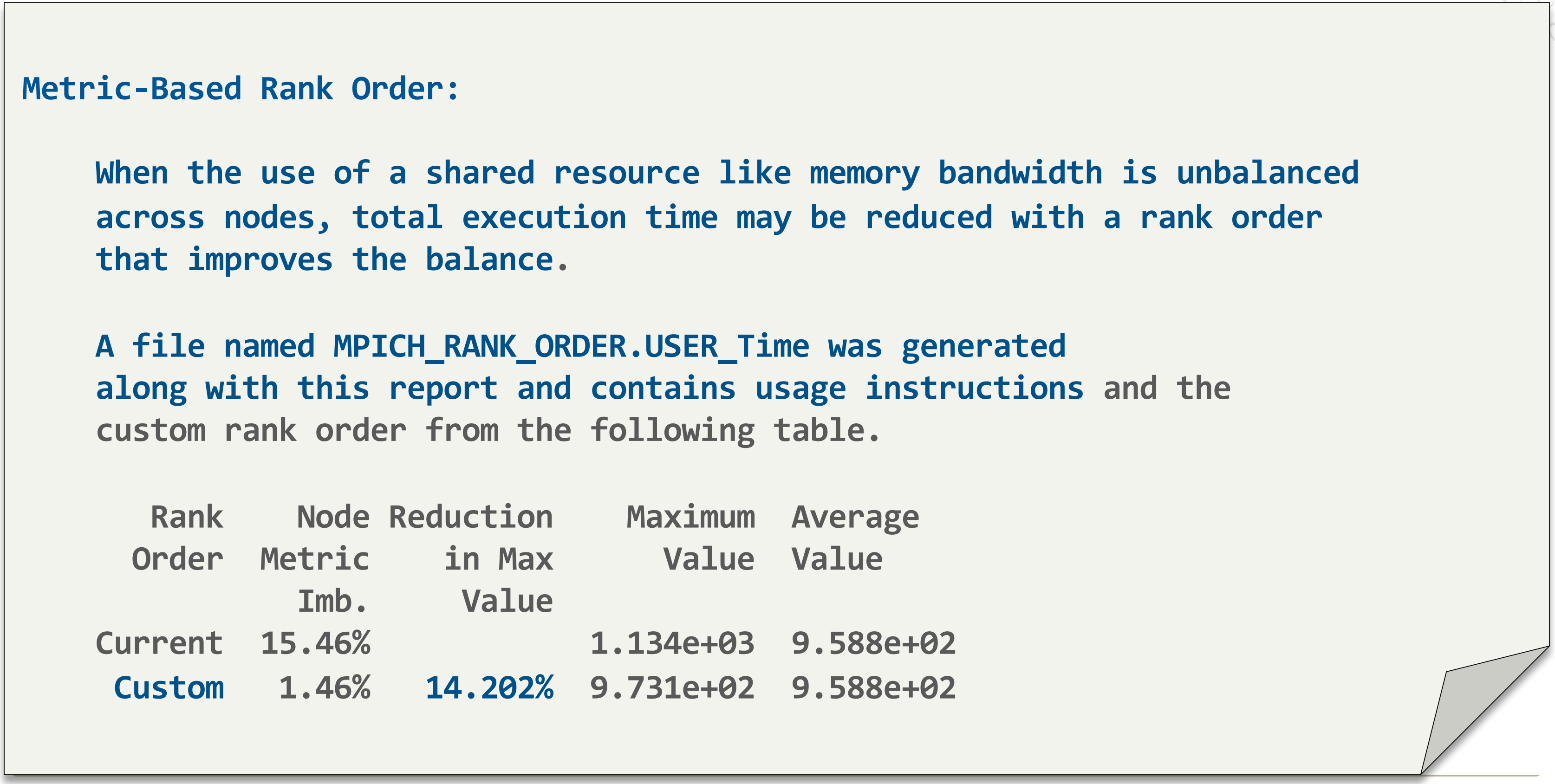


```
Table 2:  Profile by Group, Function, and Line

  Samp% |      Samp |   Imb. |  Imb. |Group
        |           |   Samp | Samp% | Function
        |           |        |       |  Source
        |           |        |       |   Line
        |           |        |       |    PE=HIDE

 100.0% | 55,605.7 |     -- |    -- |Total
|-----------------------------------------------------------------------------
|  56.5% | 31,412.8 |     -- |    -- |USER
||----------------------------------------------------------------------------
||  19.7% | 10,944.1 |     -- |    -- |create_boundary$boundary_
3|        |          |        |       | source.omp_removed/test/compile/boundary.f90
||||--------------------------------------------------------------------------
4|||   7.8% |  4,355.9 |  175.1 |  3.9% |line.265
4|||   1.8% |    977.0 |   98.0 |  9.1% |line.268
4|||   1.1% |    617.0 |   94.0 | 13.2% |line.273
4|||   2.0% |  1,133.6 |  101.4 |  8.2% |line.549
4|||   1.1% |    590.0 |   66.0 | 10.1% |line.557
||||==========================================================================
||  10.7% |  5,937.8 |     -- |    -- |get_block$blocks_
3|        |          |        |       | source.omp_removed/test/compile/blocks.f90
||||--------------------------------------------------------------------------
4|||   4.1% |  2,305.7 |  145.3 |  5.9% |line.221
4|||   1.0% |    569.5 |   77.5 | 12.0% |line.243
4|||   2.9% |  1,610.1 |  134.9 |  7.7% |line.246
```

Guidance: How Can I Learn More?

```
MPI utilization:

    The time spent processing MPI communications is relatively high.
    Functions and callsites responsible for consuming the most time can
    be found in the table generated by pat_report -O callers+src (within
    the MPI group).
```

Guidance: Reduce Shared Resource Contention



```
Metric-Based Rank Order:

    When the use of a shared resource like memory bandwidth is unbalanced
    across nodes, total execution time may be reduced with a rank order
    that improves the balance.

    A file named MPICH_RANK_ORDER.USER_Time was generated
    along with this report and contains usage instructions and the
    custom rank order from the following table.

       Rank    Node Reduction    Maximum  Average
      Order  Metric    in Max      Value  Value
               Imb.     Value
    Current  15.46%            1.134e+03  9.588e+02
     Custom   1.46%   14.202%  9.731e+02  9.588e+02
```

More In-depth Analysis and Bottleneck Detection

Types of Perftools pat_builds

- **Sampling**
  - module load perftools-base perftools
  - make <executable>
  - pat_build –O apa <executable>
    - Produces <executable>+pat
  - Execute <executable>+pat
    - Produces <filename>s.xf
  - pat_report <filename>s.xf > profile_samp
    - Produces <filename>.apa
- **Trace**
  - pat_build –O <filename>.apa
    - Produces <executable>+apa
  - Execute <executable>+apa
    - Produces <filename>t.xf
  - pat_report <filename>t.xf > profile_apa

How to Use CrayPat

- Make sure the following modules are loaded:
  - $ module load perftools-base perftools
- 2 instrumentation examples:
  - $ pat_build my_program
  - $ pat_build –u –g mpi my_program

Same sampling experiment is used when perftools-lite module is loaded

- Run application
  - $ aprun -n … my_program+pat
- Create report
  - $ pat_report my_program.xf > my_report

Predefined Trace Wrappers (-g tracegroup)

blas	Basic Linear Algebra subprograms
caf	Co-Array Fortran (Cray CCE compiler only)
hdf5	manages extremely large data collection
heap	dynamic heap
io	includes stdio and sysio groups
lapack	Linear Algebra Package
math	ANSI math
mpi	MPI
omp	OpenMP API
pthreads	POSIX threads
shmem	SHMEM
sysio	I/O system calls
system	system calls
upc	Unified Parallel C (Cray CCE compiler only)

Some may not work with Intel Compiler

For a full list, please see pat_build(1) man page

Control Data Collection with Runtime Options

- Runtime controlled through PAT_RT_XXX environment variables

- See intro_craypat(1) man page

- Examples of control
  - Enable full trace
  - Change number of data files created
  - Enable collection of CPU, network or power counter events
  - Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)

Performance Counters Overview

- **Cray supports raw counters, derived metrics and thresholds for:**
  - Processor (core and uncore)
  - Network
  - Accelerator
  - Power
- **Predefined groups**
  - Groups together counters for experiments
- **See *hwpc, nwpc, accpc, and rapl* man pages**

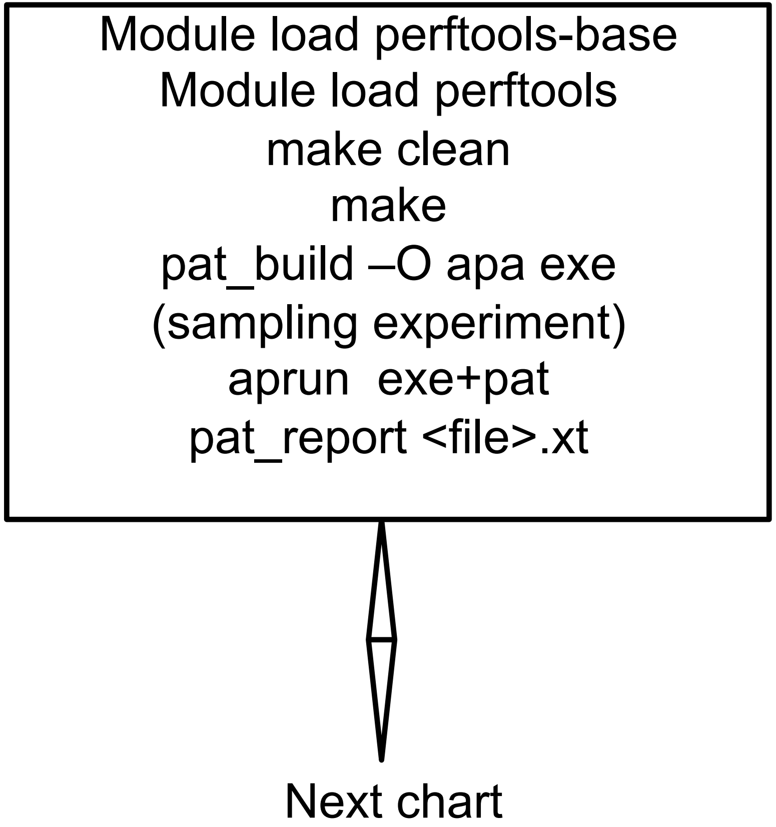
How to Get List of Events for a Processor

- **Run the following utility on a compute node:**
  - papi_native_avail

- **To collect performance counters**
  - Set PAT_RT_PERFCTR environment variable to list of events or group prior to execution

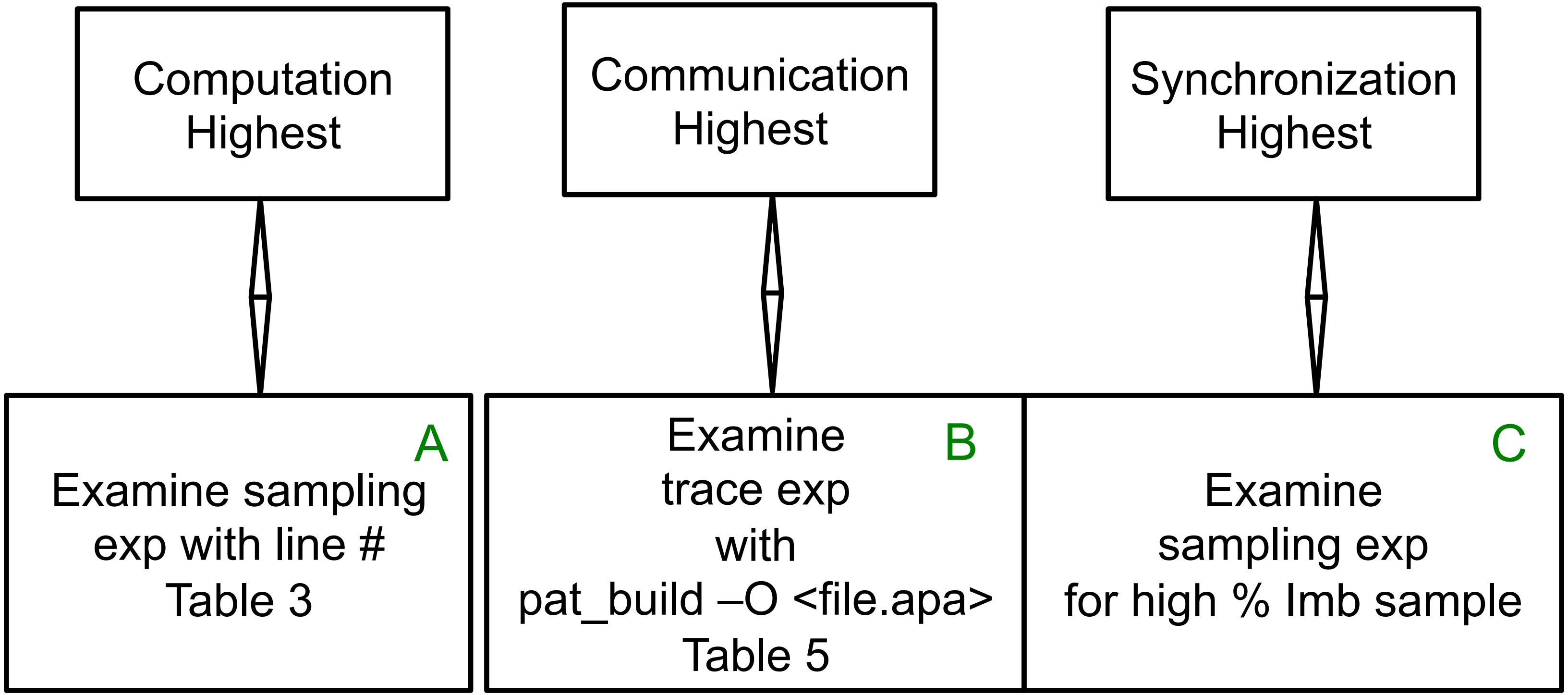
MPI Sync Time

- Measure load imbalance in programs instrumented to trace MPI functions to determine if MPI ranks arrive at collectives together
- Separates potential load imbalance from data transfer
- Sync times reported by default if MPI functions traced (pat_build –g mpi)



```
Module load perftools-base
Module load perftools
make clean
make
pat_build –O apa exe
(sampling experiment)
aprun  exe+pat
pat_report <file>.xt
```

Next chart



Computation Highest

Communication Highest

Synchronization Highest

A
Examine sampling exp with line #
Table 3

B
Examine trace exp with pat_build –O <file.apa>
Table 5

C
Examine sampling exp for high % Imb sample

If it doesn't Vectorize – fix it

Examine sampling exp with line # Table 3

```
A ||||=======================================
  ||  10.8% |   376.9 |    -- |    -- |riemann_
  3|        |         |       |       | riemann.f90
  ||||--------------------------------------------
  4|||  1.4% |    47.4 |  32.6 | 41.0% |line.77
  4|||  3.9% |   135.8 |  28.2 | 17.3% |line.78
```

```
63.  + 1----< do l = lmin, lmax
64.  + 1 2--<   do n = 1, 12
65.    1 2         pmold(l) = pmid(l)
66.    1 2         wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67.    1 2         wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68.    1 2         wlft (l) = clft(l) * sqrt(wlft(l))
69.    1 2         wrgh (l) = crgh(l) * sqrt(wrgh(l))
70.    1 2         zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71.    1 2         zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72.    1 2         zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
73.    1 2         zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
74.    1 2         umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75.    1 2         umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76.    1 2         pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l) - zlft(l))
77.    1 2         pmid (l) = max(smallp,pmid(l))
78.    1 2         if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79.    1 2-->    enddo
80.    1----> enddo
```

```
ftn-6254 ftn: VECTOR RIEMANN, File = riemann.f90, Line = 64
  A loop starting at line 64 was not vectorized because a recurrence was found
  on "pmid" at line 77.
```

If it doesn’t Vectorize – fix it

```
63.  + 1----< do l = lmin, lmax
64.  + 1 2--<   do n = 1, 12
65.    1 2        pmold(l) = pmid(l)
66.    1 2        wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67.    1 2        wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68.    1 2        wlft (l) = clft(l) * sqrt(wlft(l))
69.    1 2        wrgh (l) = crgh(l) * sqrt(wrgh(l))
70.    1 2        zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71.    1 2        zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72.    1 2        zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
73.    1 2        zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) –
                                                                      prgh(l)))
74.    1 2        umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75.    1 2        umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76.    1 2        pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) /
                                                                (zrgh(l) - zlft(l))
77.    1 2        pmid (l) = max(smallp,pmid(l))
78.    1 2        if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79.    1 2-->   enddo
80.    1----> enddo
```

```
62.    A-----<>  converged =.F.
63.  + 1------<  do n = 1, 12
64.    1 Vr2--<    do l = lmin, lmax
65.    1 Vr2        if(.not.converged(l))then
66.    1 Vr2         pmold(l) = pmid(l)
67.    1 Vr2         wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
68.    1 Vr2         wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
69.    1 Vr2         wlft (l) = clft(l) * sqrt(wlft(l))
70.    1 Vr2         wrgh (l) = crgh(l) * sqrt(wrgh(l))
71.    1 Vr2         zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
72.    1 Vr2         zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
73.    1 Vr2         zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
74.    1 Vr2         zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
75.    1 Vr2         umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
76.    1 Vr2         umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
77.    1 Vr2         pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / &
78.    1 Vr2                    (zrgh(l)-zlft(l))
79.    1 Vr2         pmid (l) = max(smallp,pmid(l))
80.    1 Vr2         if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) then
81.    1 Vr2         converged(l) = .T.
82.    1 Vr2         endif
83.    1 Vr2        endif
84.    1 Vr2-->    enddo
85.  + 1          if(all(converged(lmin:lmax)))exit
86.    1------>  enddo
```

A

If it Vectorizes, check memory utilization

```
44.   V----<   do n = nmin-1, nmax
45.   V          ar(n) = a(n) + para(n,1)*diffa(n) + para(n,2)*da(n+1) + para(n,3)*da(n)
48.   V          al(n+1) = ar(n)
49.   V---->   enddo
53.   fV---<   do n = nmin, nmax
54.   fV         onemfl= 1.0 - flat(n)
55.   fV         ar(n) = flat(n) * a(n) + onemfl * ar(n)
56.   fV         al(n) = flat(n) * a(n) + onemfl * al(n)
57.   fV--->   enddo
67.   f----< do n = nmin, nmax
68.   f        deltaa(n) = ar(n) - al(n)
69.   f        a6(n)     = 6. * (a(n) - .5 * (al(n) + ar(n)))
70.   f        scrch1(n) = (ar(n) - a(n)) * (a(n)-al(n))
71.   f        scrch2(n) = deltaa(n) * deltaa(n)
72.   f        scrch3(n) = deltaa(n) * a6(n)
73.   f----> enddo
74.
75.   Vr2--< do n = nmin, nmax
76.   Vr2      if(scrch1(n) <= 0.0) then
77.   Vr2        ar(n) = a(n)
78.   Vr2        al(n) = a(n)
79.   Vr2      endif
80.   Vr2      if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81.   Vr2      if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82.   Vr2--> enddo
83.
84.   Vr2--< do n = nmin, nmax
85.   Vr2      deltaa(n)= ar(n) - al(n)
86.   Vr2      a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
87.   Vr2--> enddo
```

```
4|||   0.1% |    3.4 |  5.6 |  62.0% |line.44
4|||   2.5% |   88.5 | 27.5 |  23.8% |line.45
4|||   0.2% |    7.1 |  8.9 |  56.0% |line.48
4|||   0.0% |    0.1 |  1.9 |  94.9% |line.49
4|||   0.3% |    9.6 | 10.4 |  52.3% |line.53
4|||   0.3% |   10.1 | 10.9 |  51.9% |line.54
4|||   0.9% |   31.3 | 14.7 |  32.0% |line.55
4|||   0.2% |    7.3 | 11.7 |  61.6% |line.56
4|||   0.0% |    0.2 |  1.8 |  90.6% |line.57
4|||   0.4% |   13.6 | 10.4 |  43.5% |line.68
4|||   0.8% |   27.2 | 13.8 |  33.9% |line.69
4|||   0.4% |   15.4 | 11.6 |  43.2% |line.70
4|||   0.4% |   13.4 | 10.6 |  44.3% |line.71
4|||   0.4% |   14.0 | 12.0 |  46.2% |line.72
4|||   0.1% |    4.8 |  8.2 |  63.5% |line.75
4|||   0.3% |   10.2 |  9.8 |  49.1% |line.76
4|||   0.6% |   19.4 | 14.6 |  43.1% |line.77
4|||   0.5% |   17.9 | 11.1 |  38.4% |line.78
4|||   1.2% |   40.6 | 16.4 |  28.8% |line.80
4|||   1.8% |   63.9 | 24.1 |  27.5% |line.81
4|||   0.0% |    0.2 |  2.8 |  92.7% |line.82
4|||   0.1% |    4.3 |  7.7 |  64.4% |line.84
4|||   0.5% |   16.0 | 12.0 |  43.2% |line.85
4|||   0.6% |   21.3 | 13.7 |  39.4% |line.86
4|||   0.0% |    0.0 |  1.0 |  95.7% |line.87
```

If it Vectorizes, check memory utilization

```
44.  V----<   do n = nmin-1, nmax
45.  V          ar(n) = a(n) + para(n,1)*diffa(n) + para(n,2)*da(n+1) + para(n,3)*da(n)
48.  V          al(n+1) = ar(n)
49.  V---->   enddo
53.  fV---<   do n = nmin, nmax
54.  fV         onemfl= 1.0 - flat(n)
55.  fV         ar(n) = flat(n) * a(n) + onemfl * ar(n)
56.  fV         al(n) = flat(n) * a(n) + onemfl * al(n)
57.  fV--->   enddo
67.  f----< do n = nmin, nmax
68.  f        deltaa(n) = ar(n) - al(n)
69.  f        a6(n)     = 6. * (a(n) - .5 * (al(n) + ar(n)))
70.  f        scrch1(n) = (ar(n) - a(n)) * (a(n)-al(n))
71.  f        scrch2(n) = deltaa(n) * deltaa(n)
72.  f        scrch3(n) = deltaa(n) * a6(n)
73.  f----> enddo
74.
75.  Vr2--< do n = nmin, nmax
76.  Vr2      if(scrch1(n) <= 0.0) then
77.  Vr2        ar(n) = a(n)
78.  Vr2        al(n) = a(n)
79.  Vr2      endif
80.  Vr2      if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
81.  Vr2      if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
82.  Vr2--> enddo
83.
84.  Vr2--< do n = nmin, nmax
85.  Vr2      deltaa(n)= ar(n) - al(n)
86.  Vr2      a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
87.  Vr2--> enddo
```

Remove unnecessary arrays and fuse loops

```
52.  Vr2--<   do n = nmin, nmax
53.  Vr2        onemfl= 1.0 - flat(n)
54.  Vr2        ar(n) = flat(n) * a(n) + onemfl * ar(n)
55.  Vr2        al(n) = flat(n) * a(n) + onemfl * al(n)
56.  Vr2      deltaa(n) = ar(n) - al(n)
57.  Vr2      a6(n)     = 6. * (a(n) - .5 * (al(n) + ar(n)))
58.  Vr2      scrch1s = (ar(n) - a(n)) * (a(n)-al(n))
59.  Vr2      scrch2s = deltaa(n) * deltaa(n)
60.  Vr2      scrch3s = deltaa(n) * a6(n)
61.  Vr2      if(scrch1s <= 0.0) then
62.  Vr2        ar(n) = a(n)
63.  Vr2        al(n) = a(n)
64.  Vr2      endif
65.  Vr2      if(scrch2s < +scrch3s) al(n) = 3. * a(n) - 2. * ar(n)
66.  Vr2      if(scrch2s < -scrch3s) ar(n) = 3. * a(n) - 2. * al(n)
67.  Vr2      deltaa(n)= ar(n) - al(n)
68.  Vr2      a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
69.  Vr2--> enddo
```

Examine
trace exp
with
pat_build –O <file.apa>
Table 5

B

```
||=========================================================
|  15.5% |  9.610204 |       -- |     -- |       361.0 |MPI
||---------------------------------------------------------
||  15.5% |  9.607925 | 0.229909 |   2.3% |       300.0 |mpi_alltoall_
```

```
Table 5:  MPI Message Stats by Caller (limited entries shown)

    MPI |   MPI Msg Bytes | MPI Msg | MsgSz | 64KiB<= |Function
    Msg |                 |   Count |   <16 |   MsgSz | Caller
  Bytes% |                |         | Count |  <1MiB  |  PE=[mmm]
         |                |         |       |  Count  |

 100.0% | 3,774,873,804.0 | 4,851.0 |  51.0 | 4,800.0 |Total
|---------------------------------------------------------------
| 100.0% | 3,774,873,600.0 | 4,800.0 |   0.0 | 4,800.0 |mpi_alltoall_
||--------------------------------------------------------------
||  66.7% | 2,516,582,400.0 | 3,200.0 |   0.0 | 3,200.0 |sweepy_
||||------------------------------------------------------------
4|||  66.7% | 2,516,582,400.0 | 3,200.0 |   0.0 | 3,200.0 |pe.0
4|||  66.7% | 2,516,582,400.0 | 3,200.0 |   0.0 | 3,200.0 |pe.128
4|||  66.7% | 2,516,582,400.0 | 3,200.0 |   0.0 | 3,200.0 |pe.255
||||============================================================
||  33.3% | 1,258,291,200.0 | 1,600.0 |   0.0 | 1,600.0 |sweepz_
||||------------------------------------------------------------
4|||  33.3% | 1,258,291,200.0 | 1,600.0 |   0.0 | 1,600.0 |pe.0
4|||  33.3% | 1,258,291,200.0 | 1,600.0 |   0.0 | 1,600.0 |pe.128
4|||  33.3% | 1,258,291,200.0 | 1,600.0 |   0.0 | 1,600.0 |pe.255
|===============================================================
```

```
22.  +          call MPI_ALLTOALL(send1,Ya2abuff_size,VH1_DATATYPE,recv1,Ya2abuff_size,
                                   VH1_DATATYPE,MPI_COMM_ROW,mpierr)
```

If Computation is load-imbalance – make sure computation is optimized

trace exp
with
pat_build –O <file.apa>
Table 2

C

```
|  10.7% |  6.636565 |       -- |     -- |       353.0 |MPI_SYNC
||--------------------------------------------------------------------------
||   4.7% |  2.888334 | 2.880515 |  99.7% |       300.0 |mpi_alltoall_(sync)
||   4.2% |  2.605623 | 2.605590 | 100.0% |         1.0 |mpi_init_(sync)
||   1.8% |  1.120161 | 1.111971 |  99.3% |        51.0 |mpi_allreduce_(sync)
|==========================================================================
```

```
|  41.8% |  5.536452 | 2.880515 |  99.7% |mpi_alltoall_(sync)
||------------------------------------------------------
||  41.8% |  5.536452 |       -- |     -- |pe.40
||   0.1% |  0.007819 |       -- |     -- |pe.96
```

```
Table 1:  Profile by Function Group and Function

  Time%  |        Time  |     Imb.  |   Imb.  |        Calls  |Group
         |              |     Time  |   Time% |               | Function
         |              |           |         |               |  PE=HIDE

 100.0%  | 61.975194  |        --  |      --  | 4,916,166.0  |Total
|------------------------------------------------------------------------
|  73.8%  | 45.728391  |        --  |      --  | 4,915,451.0  |USER
||-----------------------------------------------------------------------
||  19.4%  | 12.048791  | 1.192879  |    9.0%  |    307,200.0  |remap_
||  12.8%  |  7.930665  | 1.337482  |   14.5%  |  2,764,800.0  |parabola_
||   9.7%  |  6.006924  | 0.356937  |    5.6%  |         50.0  |sweepz_
||   9.5%  |  5.884056  | 0.335782  |    5.4%  |        100.0  |sweepy_
||   6.4%  |  3.976012  | 0.495753  |   11.1%  |    307,200.0  |riemann_
||   4.1%  |  2.530338  | 0.188650  |    7.0%  |         50.0  |sweepx2_
||   4.0%  |  2.486642  | 0.200569  |    7.5%  |         50.0  |sweepx1_
||   2.5%  |  1.523841  | 0.163506  |    9.7%  |    307,200.0  |evolve_
||   2.4%  |  1.497595  | 0.305383  |   17.0%  |    614,400.0  |paraset_
||   1.5%  |  0.923626  | 0.159765  |   14.8%  |    307,200.0  |flatten_
||   1.4%  |  0.870671  | 0.157344  |   15.4%  |    307,200.0  |states_
```

Load Imbalance on unoptimized routines

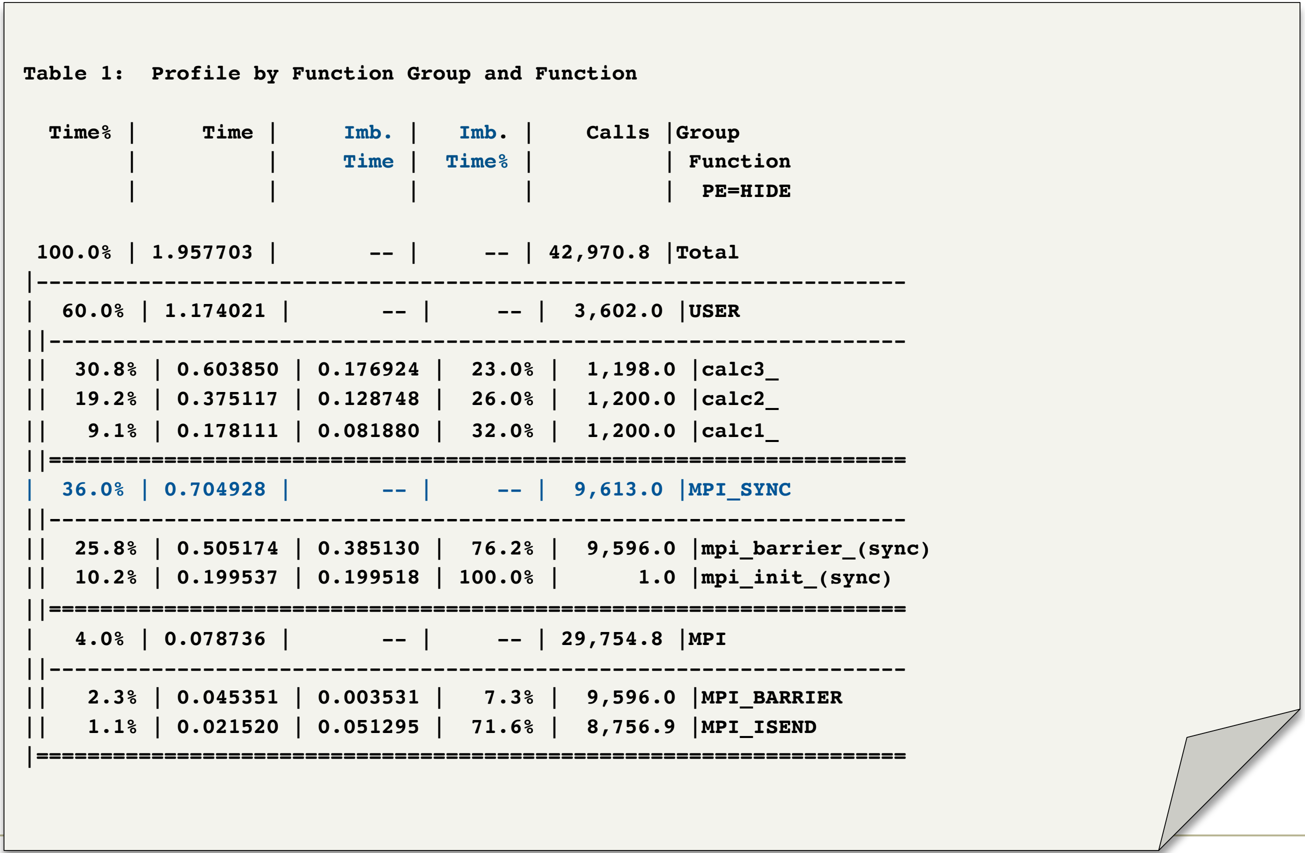
Load Imbalance on optimized routines

```
Table 1:  Profile by Function Group and Function

  Time%  |        Time  |     Imb.  |   Imb.  |        Calls  |Group
         |              |     Time  |   Time% |               | Function
         |              |           |         |               |  PE=HIDE

 100.0%  | 58.936459  |        --  |      --  | 4,916,166.0  |Total
|------------------------------------------------------------------------
|  72.3%  | 42.617036  |        --  |      --  | 4,915,451.0  |USER
||-----------------------------------------------------------------------
||  20.4%  | 12.023155  | 1.106771  |    8.5%  |    307,200.0  |remap_
||  11.0%  |  6.476465  | 1.210511  |   15.8%  |  2,764,800.0  |parabola_
||  10.1%  |  5.981124  | 0.297341  |    4.8%  |         50.0  |sweepz_
||  10.0%  |  5.887939  | 0.295370  |    4.8%  |        100.0  |sweepy_
||   4.3%  |  2.532612  | 0.159728  |    6.0%  |         50.0  |sweepx2_
||   4.2%  |  2.487010  | 0.172008  |    6.5%  |         50.0  |sweepx1_
||   4.0%  |  2.360784  | 1.441720  |   38.1%  |    307,200.0  |riemann_
||   2.6%  |  1.526218  | 0.139174  |    8.4%  |    307,200.0  |evolve_
||   2.5%  |  1.501093  | 0.268596  |   15.2%  |    614,400.0  |paraset_
||   1.6%  |  0.917283  | 0.142748  |   13.5%  |    307,200.0  |flatten_
||   1.5%  |  0.871589  | 0.141770  |   14.0%  |    307,200.0  |states_
```

Motivation for Load Imbalance Analysis

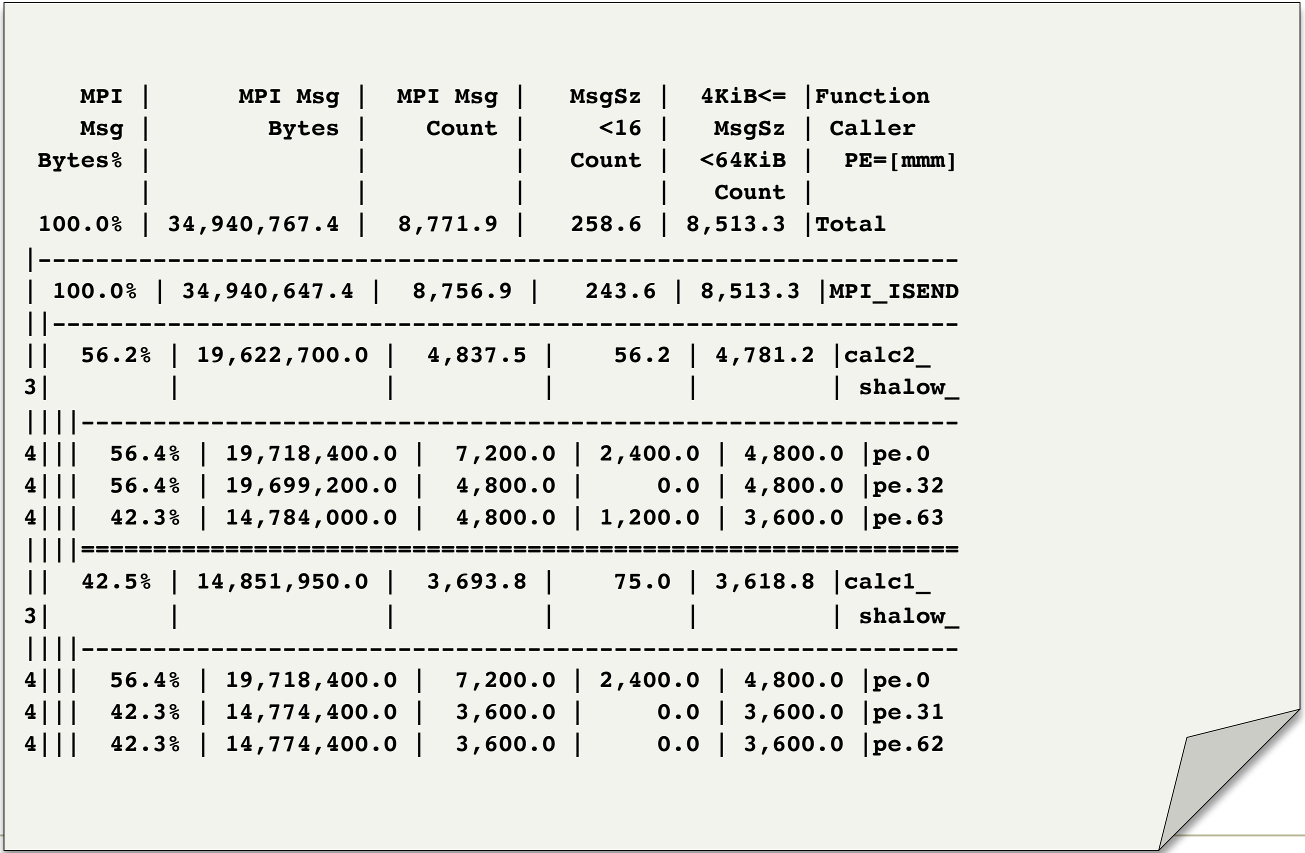
- **Increasing system software and architecture complexity**
  - Current trend in high end computing is to have systems with tens of thousands of processors
    - This is being accentuated with multi-core processors
- **Applications have to be very well balanced In order to perform at scale on these MPP systems**
  - Efficient application scaling includes a balanced use of requested computing resources
- **Desire to minimize computing resource “waste”**
  - Identify slower paths through code
  - Identify inefficient “stalls” within an application

Find Program Load Imbalance



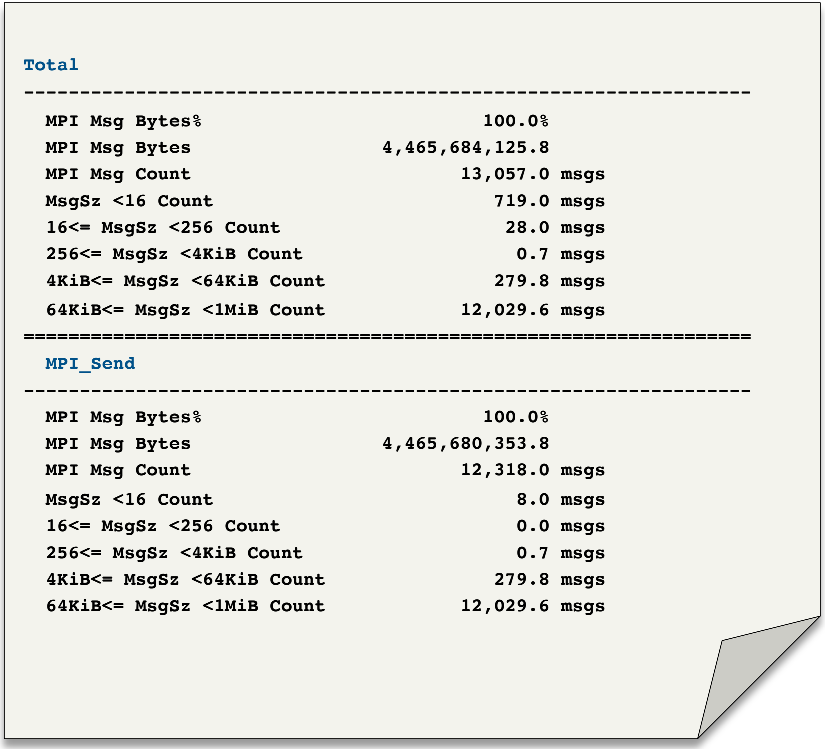
```
Table 1:  Profile by Function Group and Function

  Time% |       Time |      Imb. |   Imb. |      Calls |Group
        |            |      Time |  Time% |            | Function
        |            |           |        |            |  PE=HIDE

 100.0% | 1.957703 |        -- |     -- | 42,970.8 |Total
|-------------------------------------------------------------------
|  60.0% | 1.174021 |        -- |     -- |  3,602.0 |USER
||------------------------------------------------------------------
||  30.8% | 0.603850 | 0.176924 |  23.0% |  1,198.0 |calc3_
||  19.2% | 0.375117 | 0.128748 |  26.0% |  1,200.0 |calc2_
||   9.1% | 0.178111 | 0.081880 |  32.0% |  1,200.0 |calc1_
||==================================================================
|  36.0% | 0.704928 |        -- |     -- |  9,613.0 |MPI_SYNC
||------------------------------------------------------------------
||  25.8% | 0.505174 | 0.385130 |  76.2% |  9,596.0 |mpi_barrier_(sync)
||  10.2% | 0.199537 | 0.199518 | 100.0% |      1.0 |mpi_init_(sync)
||==================================================================
|   4.0% | 0.078736 |        -- |     -- | 29,754.8 |MPI
||------------------------------------------------------------------
||   2.3% | 0.045351 | 0.003531 |   7.3% |  9,596.0 |MPI_BARRIER
||   1.1% | 0.021520 | 0.051295 |  71.6% |  8,756.9 |MPI_ISEND
|===================================================================
```

Sort MPI Messages by Caller

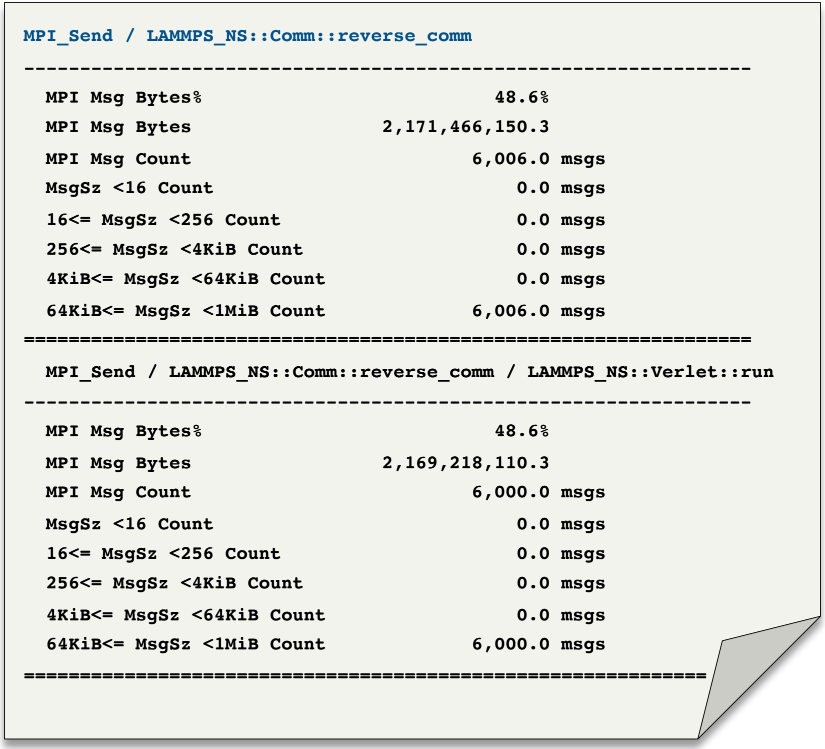


```
    MPI |       MPI Msg | MPI Msg |   MsgSz |  4KiB<= |Function
    Msg |         Bytes |   Count |     <16 |   MsgSz | Caller
 Bytes% |               |         |   Count |  <64KiB |   PE=[mmm]
        |               |         |         |   Count |
 100.0% |  34,940,767.4 | 8,771.9 |   258.6 | 8,513.3 |Total
|-----------------------------------------------------------------
| 100.0% | 34,940,647.4 |  8,756.9 |   243.6 | 8,513.3 |MPI_ISEND
||----------------------------------------------------------------
||  56.2% | 19,622,700.0 |  4,837.5 |    56.2 | 4,781.2 |calc2_
3|        |              |          |         |         | shalow_
||||--------------------------------------------------------------
4|||  56.4% | 19,718,400.0 |  7,200.0 | 2,400.0 | 4,800.0 |pe.0
4|||  56.4% | 19,699,200.0 |  4,800.0 |     0.0 | 4,800.0 |pe.32
4|||  42.3% | 14,784,000.0 |  4,800.0 | 1,200.0 | 3,600.0 |pe.63
||||==============================================================
||  42.5% | 14,851,950.0 |  3,693.8 |    75.0 | 3,618.8 |calc1_
3|        |              |          |         |         | shalow_
||||--------------------------------------------------------------
4|||  56.4% | 19,718,400.0 |  7,200.0 | 2,400.0 | 4,800.0 |pe.0
4|||  42.3% | 14,774,400.0 |  3,600.0 |     0.0 | 3,600.0 |pe.31
4|||  42.3% | 14,774,400.0 |  3,600.0 |     0.0 | 3,600.0 |pe.62
```

Analyze MPI Message Sizes



```
Total
------------------------------------------------------------------
  MPI Msg Bytes%                          100.0%
  MPI Msg Bytes                  4,465,684,125.8
  MPI Msg Count                         13,057.0 msgs
  MsgSz <16 Count                          719.0 msgs
  16<= MsgSz <256 Count                     28.0 msgs
  256<= MsgSz <4KiB Count                    0.7 msgs
  4KiB<= MsgSz <64KiB Count                279.8 msgs
  64KiB<= MsgSz <1MiB Count             12,029.6 msgs
==================================================================
  MPI_Send
------------------------------------------------------------------
  MPI Msg Bytes%                          100.0%
  MPI Msg Bytes                  4,465,680,353.8
  MPI Msg Count                         12,318.0 msgs
  MsgSz <16 Count                            8.0 msgs
  16<= MsgSz <256 Count                      0.0 msgs
  256<= MsgSz <4KiB Count                    0.7 msgs
  4KiB<= MsgSz <64KiB Count                279.8 msgs
  64KiB<= MsgSz <1MiB Count             12,029.6 msgs
```



```
MPI_Send / LAMMPS_NS::Comm::reverse_comm
------------------------------------------------------------------
  MPI Msg Bytes%                           48.6%
  MPI Msg Bytes                  2,171,466,150.3
  MPI Msg Count                          6,006.0 msgs
  MsgSz <16 Count                            0.0 msgs
  16<= MsgSz <256 Count                      0.0 msgs
  256<= MsgSz <4KiB Count                    0.0 msgs
  4KiB<= MsgSz <64KiB Count                  0.0 msgs
  64KiB<= MsgSz <1MiB Count              6,006.0 msgs
==================================================================
  MPI_Send / LAMMPS_NS::Comm::reverse_comm / LAMMPS_NS::Verlet::run
------------------------------------------------------------------
  MPI Msg Bytes%                           48.6%
  MPI Msg Bytes                  2,169,218,110.3
  MPI Msg Count                          6,000.0 msgs
  MsgSz <16 Count                            0.0 msgs
  16<= MsgSz <256 Count                      0.0 msgs
  256<= MsgSz <4KiB Count                    0.0 msgs
  4KiB<= MsgSz <64KiB Count                  0.0 msgs
  64KiB<= MsgSz <1MiB Count              6,000.0 msgs
==============================================================
```

Maximize On-node Communication by Reordering MPI ranks

When Is Rank Re-ordering Useful?

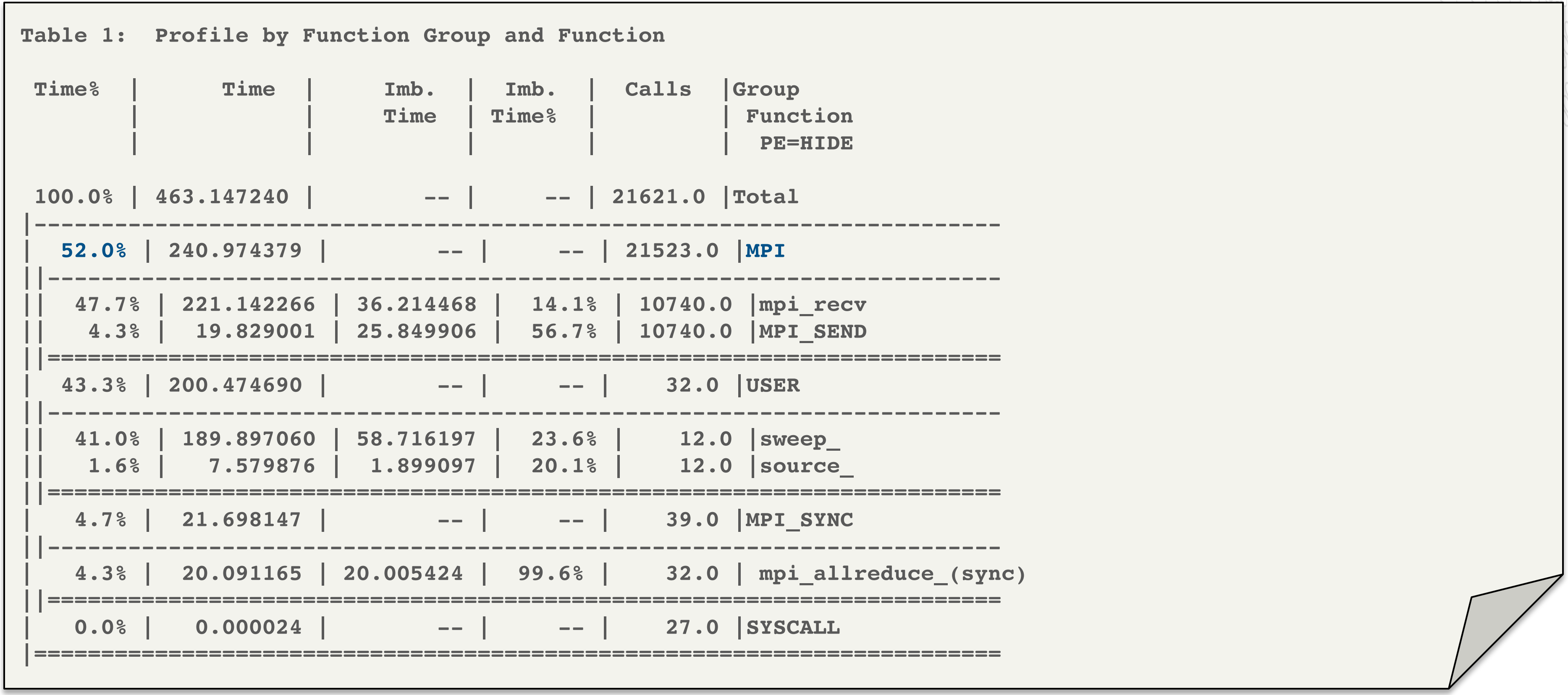
- Maximize on-node communication between MPI ranks
- Physical system topology agnostic
- Grid detection and rank re-ordering is helpful for programs with significant point-to-point communication
- Relieve on-node shared resource contention by pairing threads or processes that perform different work on the same node
  - for example: computation with off-node communication

MPI Rank Reorder – Two Interfaces Available

- **CrayPat**
  - Available with sampling or tracing
  - Include –g mpi when instrumenting program
  - Run program and let CrayPat determine if communication is dominant, detect communication pattern and suggest MPI rank order if applicable

- **grid_order utility**
  - User knows communication pattern in application and wants to quickly create a new MPI rank placement file
  - Available when perftools-base module is loaded

MPI Rank Order Observations



```
Table 1:  Profile by Function Group and Function

 Time%    |        Time  |      Imb. |   Imb. |   Calls   |Group
          |              |      Time | Time%  |           | Function
          |              |           |        |           |  PE=HIDE

 100.0% | 463.147240 |          -- |      -- | 21621.0 |Total
|-----------------------------------------------------------------------
|  52.0% | 240.974379 |          -- |      -- | 21523.0 |MPI
||----------------------------------------------------------------------
||  47.7% | 221.142266 | 36.214468 |  14.1% | 10740.0 |mpi_recv
||   4.3% |  19.829001 | 25.849906 |  56.7% | 10740.0 |MPI_SEND
||======================================================================
|  43.3% | 200.474690 |          -- |      -- |    32.0 |USER
||----------------------------------------------------------------------
||  41.0% | 189.897060 | 58.716197 |  23.6% |    12.0 |sweep_
||   1.6% |   7.579876 |  1.899097 |  20.1% |    12.0 |source_
||======================================================================
|   4.7% |  21.698147 |          -- |      -- |    39.0 |MPI_SYNC
||----------------------------------------------------------------------
|   4.3% |  20.091165 | 20.005424 |  99.6% |    32.0 | mpi_allreduce_(sync)
||======================================================================
|   0.0% |   0.000024 |          -- |      -- |    27.0 |SYSCALL
|=======================================================================
```

MPI Rank Order Observations (2)

```
MPI Grid Detection:

    There appears to be point-to-point MPI communication in a 96 X 8
    grid pattern. The 52% of the total execution time spent in MPI
    functions might be reduced with a rank order that maximizes
    communication between ranks on the same node. The effect of several
    rank orders is estimated below.

    A file named MPICH_RANK_ORDER.Grid was generated along with this
    report and contains usage instructions and the Custom rank order
    from the following table.

         Rank    On-Node    On-Node  MPICH_RANK_REORDER_METHOD
        Order   Bytes/PE  Bytes/PE%
                           of Total
                           Bytes/PE

        Custom  2.385e+09    95.55%  3
           SMP  1.880e+09    75.30%  1
          Fold  1.373e+06     0.06%  2
    RoundRobin  0.000e+00     0.00%  0
```

Auto-Generated MPI Rank Order File

```
# The 'USER_Time_hybrid' rank
order in this file targets
nodes with multi-core
# processors, based on Sent
Msg Total Bytes collected
for:
#
# Program:      /lus/
nid00023/malice/craypat/
WORKSHOP/bh2o-demo/Rank/
sweep3d/src/sweep3d
# Ap2 File:
sweep3d.gmpi-u.ap2
# Number PEs:   768
# Max PEs/Node: 16
#
# To use this file, make a
copy named MPICH_RANK_ORDER,
and set the
# environment variable
MPICH_RANK_REORDER_METHOD to
3 prior to
# executing the program.
#
0,532,64,564,32,572,96,540,8
,596,72,524,40,604,24,588
104,556,16,628,80,636,56,620
,48,516,112,580,88,548,120,6
12
1,403,65,435,33,411,97,443,9
,467,25,499,105,507,41,475
73,395,81,427,57,459,17,419,
113,491,49,387,89,451,121,48
3
6,436,102,468,70,404,38,412,
14,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,
118,420,22,452,94,388,126,48
4
129,563,193,531,161,571,225,
539,241,595,233,523,249,603,
185,555
153,587,169,627,137,635,201,
619,177,515,145,579,209,547,
217,611
7,405,71,469,39,437,103,413,
47,445,15,509,79,477,31,501
111,397,63,461,55,429,87,421
,23,493,119,389,95,453,127,4
85
134,402,198,434,166,410,230,
442,238,466,174,506,158,394,
246,474
190,498,254,426,142,458,150,
386,182,418,206,490,214,450,
222,482
128,533,192,541,160,565,232,
525,224,573,240,597,184,557,
248,605
168,589,200,517,152,629,136,
549,176,637,144,621,208,581,
216,613
5,439,37,407,69,447,101,415,
13,471,45,503,29,479,77,511
53,399,85,431,21,463,61,391,
109,423,93,455,117,495,125,4
87
2,530,34,562,66,538,98,522,1
0,570,42,554,26,594,50,602
18,514,74,586,58,626,82,546,
106,634,90,578,114,618,122,6
10
135,315,167,339,199,347,259,
307,231,371,239,379,191,331,
247,299
175,363,159,323,143,355,255,
291,207,275,183,283,151,267,
215,223
133,406,197,438,165,470,229,
414,245,446,141,478,237,502,
253,398
157,510,189,462,173,430,205,
390,149,422,213,454,181,494,
221,486
130,316,260,340,194,372,162,
348,226,308,234,380,242,332,
250,300
202,364,186,324,154,356,138,
292,170,276,178,284,210,218,
268,146
4,535,36,543,68,567,100,527,
12,599,44,575,28,559,76,607
52,591,20,631,60,639,84,519,
108,623,92,551,116,583,124,6
15
3,440,35,432,67,400,99,408,1
1,464,43,496,27,472,51,504
19,392,75,424,59,456,83,384,
107,416,91,488,115,448,123,4
80
132,401,196,441,164,409,228,
433,236,465,204,473,244,393,
188,497
252,505,140,425,212,457,156,
385,172,417,180,449,148,489,
220,481
131,534,195,542,163,566,227,
526,235,574,203,598,243,558,
187,606
251,590,211,630,179,638,139,
622,155,550,171,518,219,582,
147,614
761,660,737,652,705,668,745,
692,673,700,641,684,713,644,
753,724
729,732,681,756,721,716,764,
676,697,748,689,657,740,665,
649,708
760,528,736,536,704,560,744,
520,672,568,712,592,752,552,
640,600
728,584,680,624,720,512,696,
632,688,616,664,544,608,656,
648,576
762,659,738,651,706,667,746,
643,714,691,674,699,754,683,
730,723
722,731,763,658,642,755,739,
675,707,650,682,715,698,666,
690,747
257,345,265,313,281,305,273,
337,609,369,577,377,617,329,
513,529
545,297,633,361,625,321,585,
537,601,289,553,353,593,521,
569,561
256,373,261,341,264,349,280,
317,272,381,269,309,285,333,
277,365
352,301,320,325,288,357,328,
304,360,312,376,293,296,368,
336,344
258,338,266,346,282,314,274,
370,766,306,710,378,742,330,
678,362
646,298,750,322,718,354,758,
290,734,662,686,670,726,702,
694,654
262,375,263,343,270,311,271,
351,286,319,278,342,287,350,
279,374
294,318,358,383,359,310,295,
382,326,303,327,367,366,335,
302,334
765,661,709,663,741,653,711,
669,767,655,743,671,749,695,
679,703
677,727,751,693,647,701,717,
687,757,685,733,725,719,735,
645,759
```

Using New MPI Rank Order

- Save grid_order output to file called **MPICH_RANK_ORDER**
- `$ export MPICH_RANK_REORDER_METHOD=3`
- Run non-instrumented binary with and without new rank order to check overall wallclock time for performance improvements
- Can be used for all subsequent executions of **same job size**

Summary

- Users continue to need tools to help find critical performance bottlenecks within a program
- Cray performance tools offer functionality that reduces the time investment associated with porting and tuning applications on new and existing Cray systems

If Computation is load-imbalance – make sure to use the best MPI task mapping to nodes.

Metric-Based Rank Order:

When the use of a shared resource like memory bandwidth is unbalanced across nodes, total execution time may be reduced with a rank order that improves the balance. The metric used here for resource usage is: USER Samp

For each node, the metric values for the ranks on that node are summed. The maximum and average value of those sums are shown below for both the current rank order and a custom rank order that seeks to reduce the maximum value.

A file named MPICH_RANK_ORDER.USER_Samp was generated along with this report and contains usage instructions and the Custom rank order from the following table.

```
     Rank      Node  Reduction     Maximum   Average
    Order    Metric      in Max      Value    Value
               Imb.       Value

  Current     3.56%                3.008e+06  2.901e+06
  Custom      0.01%      3.557%   2.901e+06  2.901e+06
```

If Communication is load-imbalance – make sure to use the best MPI task mapping to nodes.

MPI Grid Detection:

There appears to be point-to-point MPI communication in a 8 X 8 X 8 grid pattern. The 16.4% of the total execution time spent in MPI functions might be reduced with a rank order that maximizes communication between ranks on the same node. The effect of several rank orders is estimated below.

A file named MPICH_RANK_ORDER.Grid was generated along with this report and contains usage instructions and the Custom rank order from the following table.

```
Rank Order      On-Node      On-Node  MPICH_RANK_REORDER_METHOD
              Bytes/PE    Bytes/PE%
                           of Total
                           Bytes/PE

    Custom   3.396e+12       74.82%      3
RoundRobin   3.026e+12       66.67%      0
       SMP   2.680e+12       59.04%      1
      Fold   1.700e+12       37.45%      2
```

A pointer for looking at load imbalance

- When running on a large number of MPI tasks, the load imbalance is averaged over all the tasks. This can result in an obscure situation where a seemingly small load imbalance can hide a performance bottleneck.
- For example, in a run of the NIF problem we had a bad load imbalance and the culprit was
- || 0.4% | 82.6 | 195.4 | 70.8% |hypre_BinarySearch
  - This was run on 128 MPI tasks (2 nodes); however, while the average load imbalance was an average of 82.6 samples (.826 seconds) some of the MPI tasks took significantly longer. By optimizing hypre-BinarySearch we were able to get a nice performance boost.

Memory Analysis Tool

- **First load modules**
  - module load perftools-base perftools-lite-hbm
  - make <executable>
  - Execute <executable>
  - After execution you will see a <filename>.rpt and/or information attached to stdout

Will not work
with Intel Compiler

- **The memory analysis tool can be run on KNL or Haswell**
  - Better information is obtained on Haswell due to better hardware counters
    - The tool understands that the target system is KNL, so Haswell hardware counters are used to identify issues on KNL

```
#################################################################
#                                                               #
#            CrayPat-lite Performance Statistics                #
#                                                               #
#################################################################

CrayPat/X:  Version 6.4.6 Revision 7d0d87c  02/20/17 09:52:37
Experiment:                  lite  lite/hbm
Number of PEs (MPI ranks):    512
Numbers of PEs per Node:       32  PEs on each of  16  Nodes
Numbers of Threads per PE:      1
Number of Cores per Socket:    18
Execution start time:  Thu Feb 23 09:50:22 2017
System name and speed:  kay  2101 MHz (approx)
Intel broadwell CPU  Family:  6  Model: 79  Stepping:  1


Avg Process Time:     246.96 secs
High Memory:       596,629.9 MBytes 1,165.3 MBytes per PE

Sampling overflow event and threshold:
   MEM_LOAD_UOPS_RETIRED:HIT_LFB:precise=2   400,009
```

Will not work
with Intel Compiler

```
Table 1:  HBM Objects Sorted by Weighted Sample

  Object | Object |   Object | Object |    Object | Object Location
  Sample | Sample |   Sample |    Max | Max Size |  PE=HIDE
 Weight% | Count% |    Count | Active |  (MBytes) |

  100.0% | 100.0% | 25,751.0 |  581.4 | 1,611.566 | Total
|-----------------------------------------------------------------------------
|   12.8% |   7.2% | 1,841.6 |    1.0 |    43.875 | yspecies@allocate@../source/f90_files/solve/init_field.f90@line.8
|   11.5% |   9.0% | 2,320.0 |    1.0 |   141.750 | q@allocate@../source/f90_files/solve/init_field.f90@line.83
|   11.2% |  18.9% | 4,854.1 |    1.0 |     0.276 | $$_rf@local@<unknown>@line.0
|   10.0% |   8.2% | 2,107.5 |    1.0 |   131.625 | diffflux@local@../source/f90_files/solve/rhsf.f90@line.101
|    9.9% |  16.5% | 4,260.7 |    1.0 |     0.276 | $$_rb@local@<unknown>@line.0
|    8.3% |   3.1% |   804.8 |    1.0 |   131.625 | grad_ys@local@../source/f90_files/solve/rhsf.f90@line.101
|    6.4% |   2.6% |   666.6 |    1.0 |     0.844 | phi@allocate@../source/modules/mixfrac_m.f90@line.166
|    5.9% |   2.8% |   723.8 |    1.0 |   141.750 | tmmp@local@../source/f90_files/solve/rhsf.f90@line.89
|    4.2% |   6.8% | 1,754.1 |    1.0 |     2.531 | u@allocate@../source/f90_files/solve/init_field.f90@line.83
|    2.2% |   3.8% |   967.4 |    1.0 |     0.051 | $$_dif@local@<unknown>@line.0
|    2.0% |   3.3% |   848.9 |    1.0 |     0.051 | $$_c@local@<unknown>@line.0
|    1.4% |   0.7% |   186.2 |    1.0 |    47.250 | q_err@allocate@../source/modules/erk_m.f90@line.463
|    1.2% |   2.0% |   514.7 |    1.0 |     0.051 | $$_x@local@<unknown>@line.0
|    1.1% |   0.4% |   103.3 |    1.0 |    43.875 | h_spec@local@../source/f90_files/solve/rhsf.f90@line.111
|=============================================================================
```

Will not work
with Intel Compiler

```
Table 2:  HBM Objects Sorted by Sample Count

 Object |  Object |   Object | Object |    Object | Object Location
 Sample |  Sample |   Sample |    Max |  Max Size |  PE=HIDE
 Count% | Weight% |    Count | Active |  (MBytes) |

 100.0% |  100.0% | 25,751.0 |  581.4 | 1,611.566 | Total
|-----------------------------------------------------------------------------
|  18.9% |   11.2% |  4,854.1 |    1.0 |     0.276 | $$_rf@local@<unknown>@line.0
|  16.5% |    9.9% |  4,260.7 |    1.0 |     0.276 | $$_rb@local@<unknown>@line.0
|   9.0% |   11.5% |  2,320.0 |    1.0 |   141.750 | q@allocate@../source/f90_files/solve/init_field.f90@line.83
|   8.2% |   10.0% |  2,107.5 |    1.0 |   131.625 | diffflux@local@../source/f90_files/solve/rhsf.f90@line.101
|   7.2% |   12.8% |  1,841.6 |    1.0 |    43.875 | yspecies@allocate@../source/f90_files/solve/init_field.f90@line.8
|   6.8% |    4.2% |  1,754.1 |    1.0 |     2.531 | u@allocate@../source/f90_files/solve/init_field.f90@line.83
|   3.8% |    2.2% |    967.4 |    1.0 |     0.051 | $$_dif@local@<unknown>@line.0
|   3.3% |    2.0% |    848.9 |    1.0 |     0.051 | $$_c@local@<unknown>@line.0
|   3.1% |    8.3% |    804.8 |    1.0 |   131.625 | grad_ys@local@../source/f90_files/solve/rhsf.f90@line.101
|   2.8% |    5.9% |    723.8 |    1.0 |   141.750 | tmmp@local@../source/f90_files/solve/rhsf.f90@line.89
|   2.6% |    6.4% |    666.6 |    1.0 |     0.844 | phi@allocate@../source/modules/mixfrac_m.f90@line.166
|   2.1% |    0.7% |    549.4 |    1.0 |     0.021 | sqrtq@local@../source/modules/computeCoefficients_r.f90@line.122
|   2.0% |    1.2% |    514.7 |    1.0 |     0.051 | $$_x@local@<unknown>@line.0
|   1.6% |    0.9% |    403.0 |    1.0 |     0.051 | $$_y@local@<unknown>@line.0
|   1.3% |    0.7% |    334.3 |    1.0 |     0.051 | $$_xs@local@<unknown>@line.0
|   1.1% |    0.3% |    283.6 |    1.0 |     0.051 | $$_rmcwrk1@local@<unknown>@line.0
|   1.0% |    0.9% |    268.9 |    1.0 |    43.875 | diffusion@local@../source/f90_files/solve/rhsf.f90@line.147
|   1.0% |    0.6% |    246.3 |    1.0 |     0.051 | $$_xxwt@local@<unknown>@line.0
|=============================================================================
```

Will not work
with Intel Compiler

```
Table 4:  Profile by Function

  Samp% |      Samp |  Imb. |  Imb. | MEM_LOAD_UOPS_RETIRED | RESOURCE_STALLS | Group
        |           |  Samp | Samp% |    :HIT_LFB:precise=2 |            :ALL |  Function=[MAX10]
        |           |       |       |                       |                 |   PE=HIDE

 100.0% | 26,420.2 |    -- |    -- |        10,569,018,266 | 407,403,254,579 | Total
|-----------------------------------------------------------------------------
|  96.9% | 25,594.0 |    -- |    -- |        10,238,474,892 | 385,090,314,646 | USER
||----------------------------------------------------------------------------
||  45.7% | 12,078.5 | 309.5 |  2.5% |         4,831,543,082 | 111,983,149,693 | reaction_rate_vec_.LOOP@li.347
||  13.1% |  3,460.0 | 430.0 | 11.1% |         1,384,031,140 |  24,253,285,440 | rhsf_.LOOP@li.924
||   6.1% |  1,610.0 | 116.0 |  6.7% |           644,008,240 |  14,874,097,115 | computecoefficients_r_.LOOP@li.277
||   4.9% |  1,291.7 |  70.3 |  5.2% |           516,699,907 |   6,342,301,352 | computecoefficients_r_.LOOP@li.157
||   4.2% |  1,115.9 |  33.1 |  2.9% |           446,392,856 |  46,282,065,332 | rhsf_.LOOP@li.626
||   4.0% |  1,064.6 | 152.4 | 12.6% |           425,838,487 |  30,080,185,807 | derivative_x_bnds_buff_r_
||   3.5% |    920.1 |  18.9 |  2.0% |           368,067,656 |  24,815,310,694 | calc_primary_vars_.LOOP@li.52
||   3.0% |    793.4 |  17.6 |  2.2% |           317,354,015 |  27,640,713,430 | integrate_.LOOP@li.121
||   2.2% |    582.5 |  58.5 |  9.2% |           232,991,961 |  12,088,490,382 | rhsf_.LOOP@li.1742
||============================================================================
|   3.1% |    817.5 |    -- |    -- |           327,059,702 |  21,968,897,774 | ETC
||----------------------------------------------------------------------------
||   2.1% |    544.6 | 124.4 | 18.6% |           217,925,216 |  19,260,243,388 | _cray_mpi_memcpy_snb
|=============================================================================
```

Will not work
with Intel Compiler

```
Table 1:  Profile by Function

  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  PE=HIDE

 100.0% | 56,875.0 |      -- |    -- |Total
|-----------------------------------------------------------------------
|  76.2% | 43,356.4 |      -- |    -- |USER
||----------------------------------------------------------------------
||  23.4% | 13,307.7 |   912.3 |  6.4% |reaction_rate_vec_.LOOP@li.349
||   9.2% |  5,238.2 |   943.8 | 15.3% |rhsf_.LOOP@li.626
||   5.5% |  3,131.3 |   212.7 |  6.4% |rhsf_.LOOP@li.1643
||   5.4% |  3,045.9 |    99.1 |  3.2% |computecoefficients_r_.LOOP@li.277
||   5.3% |  3,012.7 |   319.3 |  9.6% |rhsf_.LOOP@li.1687
||   4.6% |  2,606.5 |   252.5 |  8.8% |calc_primary_vars_.LOOP@li.49
||   3.7% |  2,110.6 | 1,192.4 | 36.2% |rhsf_.LOOP@li.924
||   2.5% |  1,433.7 |   110.3 |  7.2% |computecoefficients_r_.LOOP@li.157
||   2.4% |  1,375.5 |   402.5 | 22.7% |rhsf_.LOOP@li.454
||   2.2% |  1,231.2 |   689.8 | 36.0% |integrate_.LOOP@li.121
||   2.1% |  1,201.5 |   348.5 | 22.5% |derivative_x_bnds_buff_r_
||   1.5% |    863.5 |   205.5 | 19.3% |rhsf_.LOOP@li.1742
||======================================================================
|  18.8% | 10,696.5 |      -- |    -- |MPI
||----------------------------------------------------------------------
||  12.4% |  7,049.7 | 3,691.3 | 34.4% |MPI_WAITANY
||   5.1% |  2,876.8 | 3,602.2 | 55.7% |mpi_waitall
||======================================================================
|   4.9% |  2,799.9 |      -- |    -- |ETC
||----------------------------------------------------------------------
||   3.2% |  1,840.0 |   174.0 |  8.7% |_cray_ALOG10
||   1.5% |    874.3 |    90.7 |  9.4% |_EXP_Z
```

Will not work with Intel Compiler

```
Table 5:  Profile by Group, Function, and Line

  Samp% |      Samp |  Imb. |  Imb. | MEM_LOAD_UOPS_RETIRED | RESOURCE_STALLS | Group
        |           |  Samp | Samp% |    :HIT_LFB:precise=2 |            :ALL |  Function=[MAX10]
        |           |       |       |                       |                 |   Source
        |           |       |       |                       |                 |    Line
        |           |       |       |                       |                 |     PE=HIDE

 100.0% | 26,420.2 |    -- |    -- |        10,569,018,266 | 407,403,254,579 | Total
|--------------------------------------------------------------------------------
|  96.9% | 25,594.0 |    -- |    -- |        10,238,474,892 | 385,090,314,646 | USER
||-------------------------------------------------------------------------------
||  45.7% | 12,078.5 |    -- |    -- |        4,831,543,082 | 111,983,149,693 | reaction_rate_vec_.LOOP@li.347
3|        |          |       |       |                      |                 |  getrates.f
||||-----------------------------------------------------------------------------
4|||   1.0% |    261.3 |  50.7 | 16.3% |         104,503,914 |   2,421,756,549 | line.486
4|||   1.1% |    299.3 |  65.7 | 18.0% |         119,716,756 |   2,733,588,335 | line.3479
4|||   1.0% |    274.5 | 155.5 | 36.2% |         109,794,658 |   2,521,794,348 | line.5001
||||=============================================================================
||  13.1% |  3,460.0 |    -- |    -- |        1,384,031,140 |  24,253,285,440 | rhsf_.LOOP@li.924
3|        |          |       |       |                      |                 |  rhsf.f90
||||-----------------------------------------------------------------------------
4|||   5.5% |  1,443.5 | 211.5 | 12.8% |         577,416,117 |  10,080,883,003 | line.933
4|||   3.7% |    971.9 | 172.1 | 15.1% |         388,757,966 |   6,818,669,249 | line.938
4|||   4.0% |  1,044.5 | 200.5 | 16.1% |         417,807,057 |   7,352,846,349 | line.943
||||=============================================================================
||   6.1% |  1,610.0 |    -- |    -- |          644,008,240 |  14,874,097,115 | computecoefficients_r_.LOOP@li.277
3|        |          |       |       |                      |                 |  computeCoefficients_r.f90
4|   1.0% |    271.2 |  54.8 | 16.9% |          108,464,159 |   2,504,937,154 |   line.374
||   4.9% |  1,291.7 |    -- |    -- |          516,699,907 |   6,342,301,352 | computecoefficients_r_.LOOP@li.157
```

Will not work
with Intel Compiler

Excerpt from getrates.f

Will not work
with Intel Compiler

```
3478.     M m V         C     oh
3479.     M m V               DDOT=+RF(I,3)+RB(I,4)+RF(I,5)+RF(I,6)+RB(I,7)+RB(I,8)+RB(I,9)
3480.     M m V              *+RB(I,9)+RF(I,10)+RF(I,12)+RB(I,15)+RF(I,19)+RB(I,20)+RB(I,21)
3481.     M m V              *+RB(I,22)+RB(I,24)+RF(I,30)+RB(I,32)+RF(I,33)+RB(I,40)+RB(I,45)
3482.     M m V              *+RB(I,45)+RF(I,47)+RF(I,49)+RF(I,49)+RB(I,50)+RF(I,55)+RB(I,57)
3483.     M m V              *+RF(I,58)+RB(I,59)+RF(I,61)+RB(I,64)+RB(I,68)+RF(I,70)+RF(I,71)
3484.     M m V              *+RF(I,73)+RF(I,82)+RF(I,90)+RB(I,92)+RF(I,102)+RF(I,104)+RF(I,105)
3485.     M m V              *+RF(I,106)+RB(I,112)+RF(I,128)+RB(I,136)+RF(I,138)+RF(I,139)
3486.     M m V              *+RB(I,152)+RB(I,156)+RF(I,160)+RF(I,167)+RB(I,169)+RB(I,176)
3487.     M m V              *+RF(I,178)+RB(I,182)+RF(I,184)+RF(I,188)+RF(I,189)+RF(I,190)
3488.     M m V              *+RB(I,197)+RF(I,202)+RF(I,208)+RF(I,211)+RF(I,221)+RB(I,222)
3489.     M m V              *+RF(I,224)+RB(I,226)+RF(I,232)+RF(I,236)+RB(I,250)+RF(I,252)
3490.     M m V              *+RF(I,253)+RF(I,282)+RF(I,283)
```

Excerpt from rhsf.f90

```
924.  + M  m-----------------<   SPECIESX: do n=1,n_spec-1
925.    M  m                   #ifdef GPU
926.    M  D                   !$acc loop vector  private(i, ml, mu,itmp)
927.    M  m                   #endif
928.  + M  m 2---------------<   do i = 1, nx*ny*nz, ms
929.    M  m 2                      ml = i
930.    M  m 2                      mu =  min(i+ms-1, nx*ny*nz)
931.    M  m 2                      if (vary_in_x==1) then
932.    M  m 2                      itmp = 15 + n
933.  + M  m 2 r4-----------<>         tmmp(ml:mu,1,1,itmp) = -q(ml:mu,1,1,5+n)*u(ml:mu,1,1,1) - diffFlux(ml:mu,1,1,
934.    M  m 2
935.    M  m 2                      endif
936.    M  m 2                      if (vary_in_y==1) then
937.    M  m 2                      itmp = 15 + (n_spec-1) + n
938.  + M  m 2 r4-----------<>         tmmp(ml:mu,1,1,itmp) = -q(ml:mu,1,1,5+n)*u(ml:mu,1,1,2) - diffFlux(ml:mu,1,1,
939.    M  m 2
940.    M  m 2                      endif
941.    M  m 2                      if (vary_in_z==1) then
942.    M  m 2                      itmp = 15 + 2*(n_spec-1) + n
943.  + M  m 2 r4-----------<>         tmmp(ml:mu,1,1,itmp) = -q(ml:mu,1,1,5+n)*u(ml:mu,1,1,3) - diffFlux(ml:mu,1,1,
944.    M  m 2                      endif
945.    M  m 2--------------->    enddo
946.    M  m----------------->   enddo SPECIESX
```

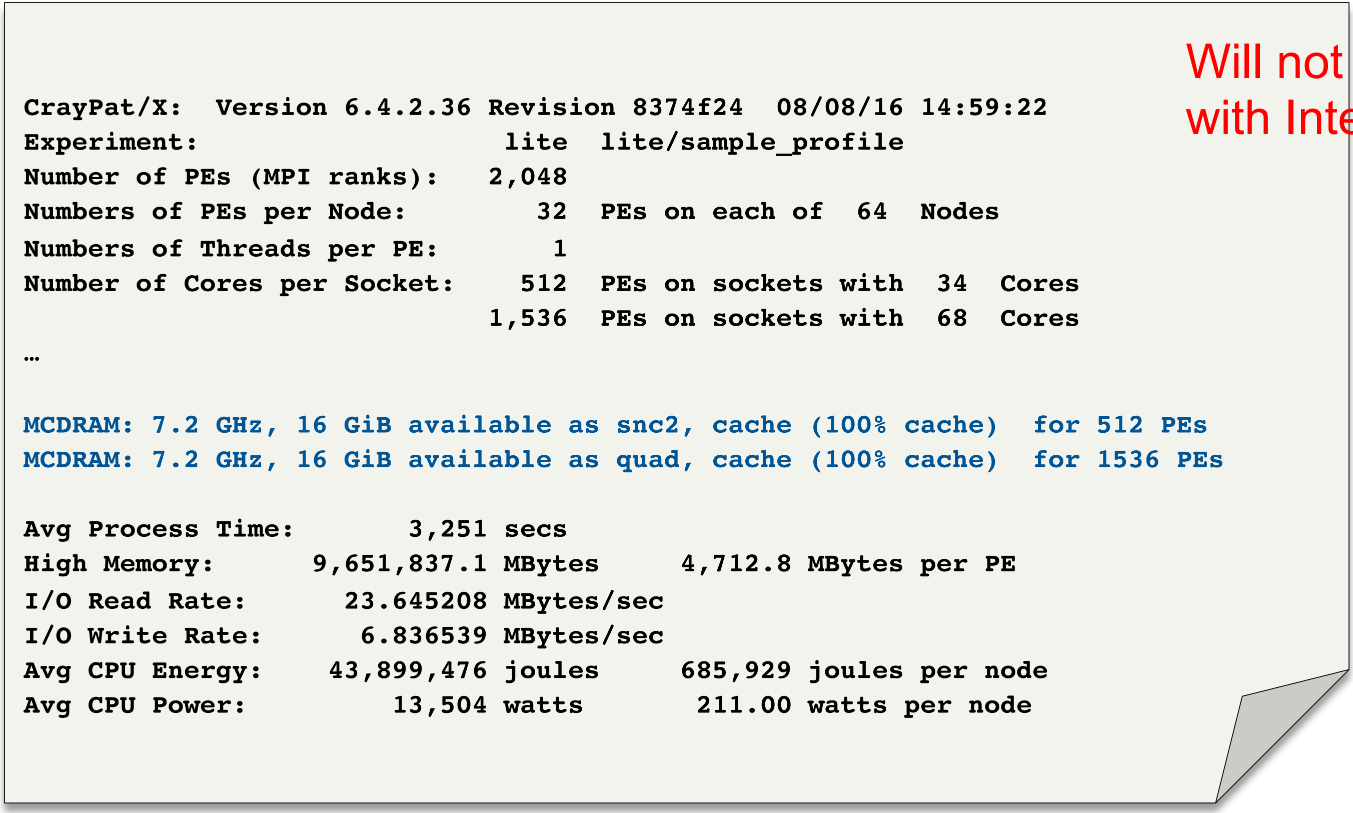
Will not work
with Intel Compiler

Tips When Running on KNL with Directives

Will not work
with Intel Compiler

- **Check for MCDRAM allocations**
  - Set CCE's CRAYMEM_DEBUG environment variable to '1' or '2'
  - Look for posix_memalign(allocator-"bandwidth" size=…) messages
- **Try running in split mode (25% cache) to reduce effect of TLB misses since there is no L3 cache**
  - If no performance gain over 100%, try flat
- **Don't allocate local variables within a routine to MCDRAM**
  - Convert to global first

MCDRAM Configuration Information



```
CrayPat/X:  Version 6.4.2.36 Revision 8374f24  08/08/16 14:59:22
Experiment:                   lite  lite/sample_profile
Number of PEs (MPI ranks):   2,048
Numbers of PEs per Node:        32  PEs on each of  64  Nodes
Numbers of Threads per PE:       1
Number of Cores per Socket:    512  PEs on sockets with  34  Cores
                             1,536  PEs on sockets with  68  Cores
…

MCDRAM: 7.2 GHz, 16 GiB available as snc2, cache (100% cache)  for 512 PEs
MCDRAM: 7.2 GHz, 16 GiB available as quad, cache (100% cache)  for 1536 PEs

Avg Process Time:       3,251 secs
High Memory:      9,651,837.1 MBytes     4,712.8 MBytes per PE
I/O Read Rate:      23.645208 MBytes/sec
I/O Write Rate:      6.836539 MBytes/sec
Avg CPU Energy:    43,899,476 joules     685,929 joules per node
Avg CPU Power:         13,504 watts       211.00 watts per node
```

Will not work with Intel Compiler

Per-numanode Memory High Water Mark

Will not work
with Intel Compiler

-
  ```
  $ module load perftools-lite
  ```
- Build and Run program:

  -
    ```
    1) $ aprun -m1800m -n 2048 -N 16 -d 1 ./cpmd.x
    ```

  -
    ```
    2) $ aprun -m1800m -n 2048 -N 16 -d 1 \
              numactl --preferred=1 ./cpmd.x
    ```

  -
    ```
    3) $ aprun -m1800m -n 2048 -N 64 -d 1 \
              numactl --preferred=1 ./cpmd.x
    ```
- Run pat_report to get memory high water mark breakdown

  -
    ```
    $ pat_report -O himem cpmd.x.*ap2 > rpt
    ```

1) Flat Mode, No Allocation

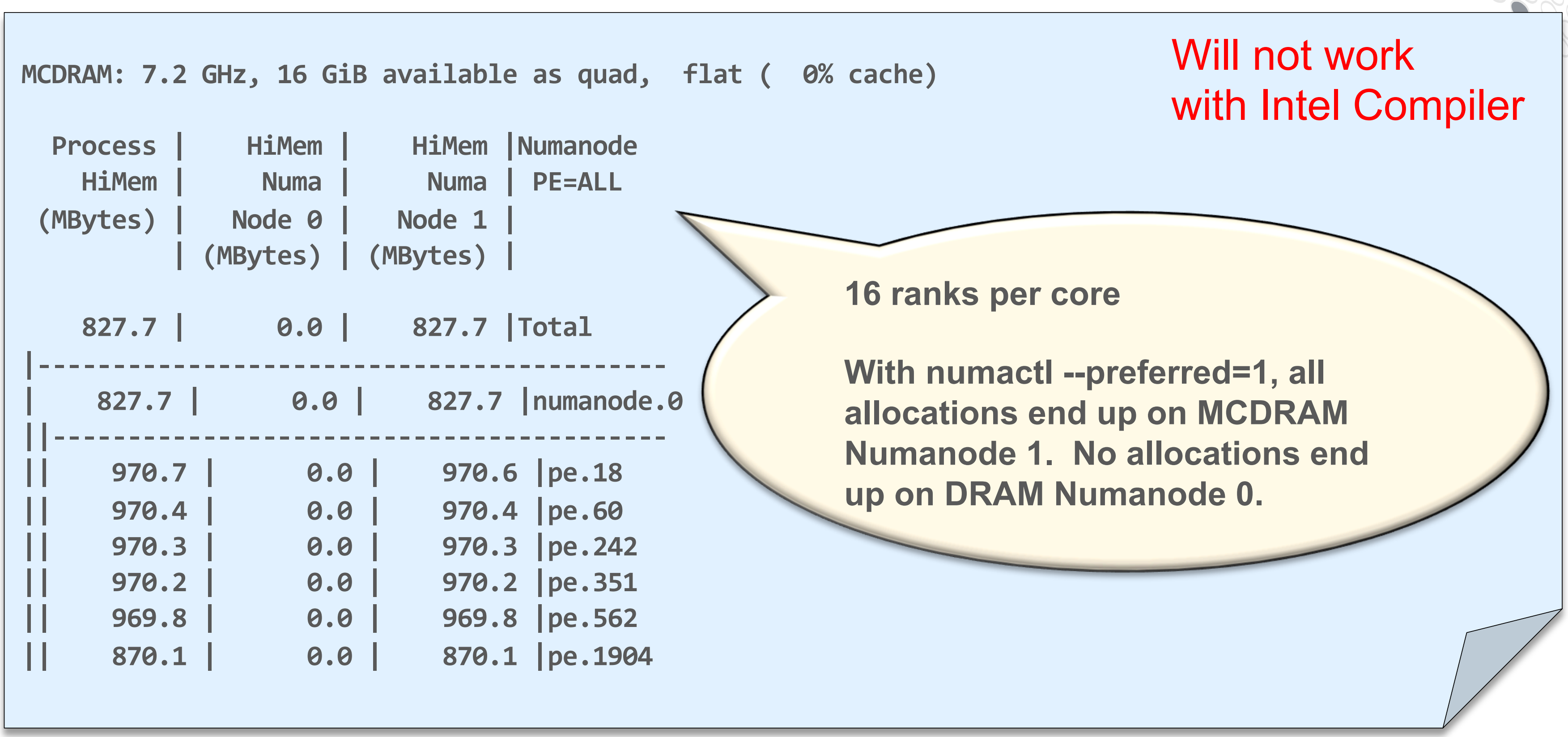
Will not work with Intel Compiler

```
MCDRAM: 7.2 GHz, 16 GiB available as quad,  flat (  0% cache)

  Process |     HiMem |Numanode
    HiMem |      Numa | PE=ALL
 (MBytes) |    Node 0 |
          |  (MBytes) |

    827.3 |     827.3 |Total
|-------------------------------
|    827.3 |     827.3 |numanode.0
||------------------------------
||    870.5 |     870.5 |pe.1904
||    870.0 |     870.0 |pe.2000
||    869.8 |     869.8 |pe.1776
||    868.6 |     868.6 |pe.1008
||    868.1 |     868.1 |pe.1264
```

16 ranks per core

No allocation to MCDRAM, only to DRAM so one Numanode is used

2) Flat Mode, Full Allocation



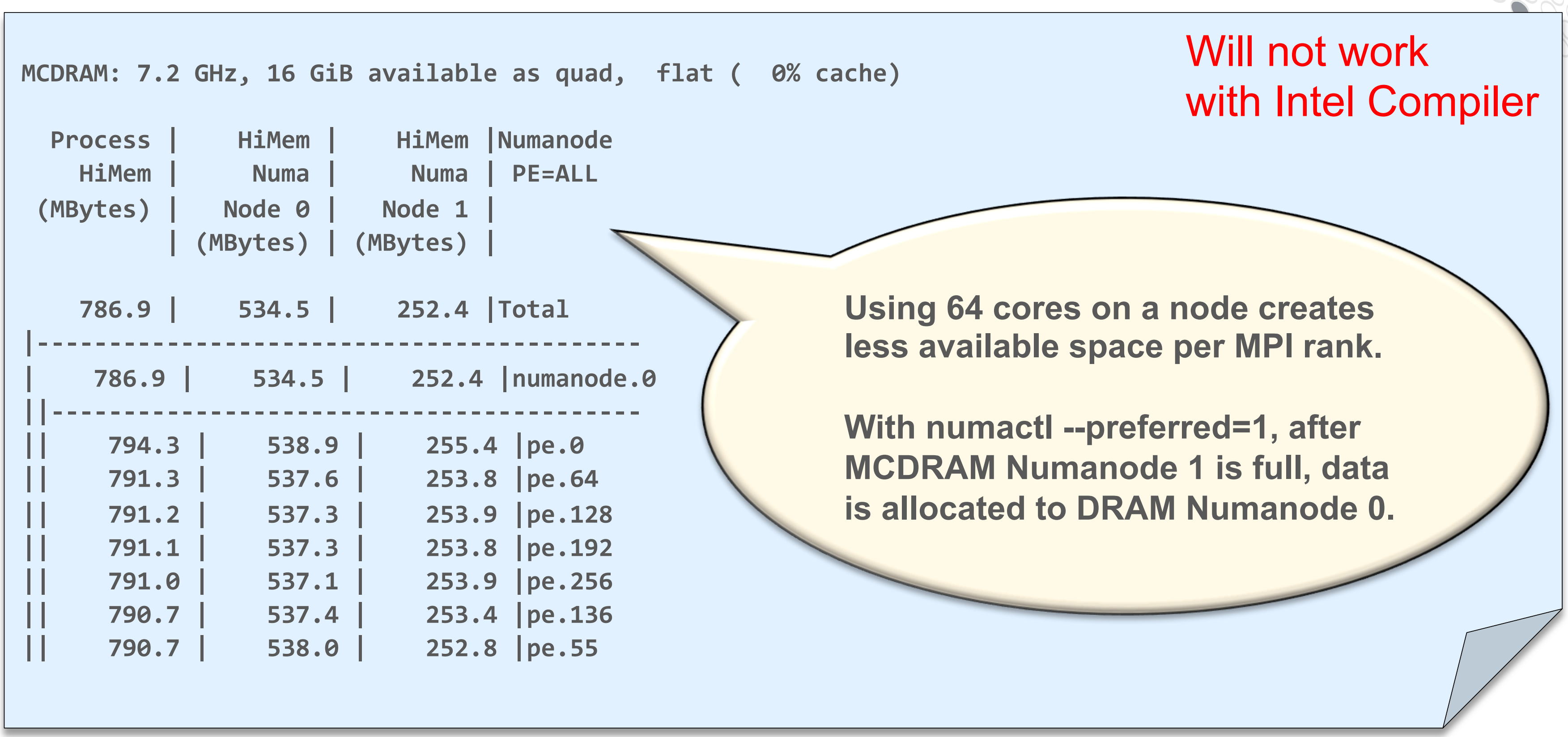
```
MCDRAM: 7.2 GHz, 16 GiB available as quad,  flat (  0% cache)

  Process |     HiMem |     HiMem |Numanode
    HiMem |      Numa |      Numa | PE=ALL
 (MBytes) |    Node 0 |    Node 1 |
          |  (MBytes) |  (MBytes) |

    827.7 |       0.0 |     827.7 |Total
|-----------------------------------------
|    827.7 |       0.0 |     827.7 |numanode.0
||----------------------------------------
||    970.7 |       0.0 |     970.6 |pe.18
||    970.4 |       0.0 |     970.4 |pe.60
||    970.3 |       0.0 |     970.3 |pe.242
||    970.2 |       0.0 |     970.2 |pe.351
||    969.8 |       0.0 |     969.8 |pe.562
||    870.1 |       0.0 |     870.1 |pe.1904
```

Will not work with Intel Compiler

16 ranks per core

With numactl --preferred=1, all allocations end up on MCDRAM Numanode 1. No allocations end up on DRAM Numanode 0.

3) Flat Mode, Partial Allocation



Will not work with Intel Compiler

```
MCDRAM: 7.2 GHz, 16 GiB available as quad,  flat (  0% cache)

  Process |     HiMem |     HiMem |Numanode
    HiMem |      Numa |      Numa | PE=ALL
 (MBytes) |    Node 0 |    Node 1 |
          |  (MBytes) |  (MBytes) |

    786.9 |     534.5 |     252.4 |Total
|------------------------------------------
|    786.9 |     534.5 |     252.4 |numanode.0
||-----------------------------------------
||    794.3 |     538.9 |     255.4 |pe.0
||    791.3 |     537.6 |     253.8 |pe.64
||    791.2 |     537.3 |     253.9 |pe.128
||    791.1 |     537.3 |     253.8 |pe.192
||    791.0 |     537.1 |     253.9 |pe.256
||    790.7 |     537.4 |     253.4 |pe.136
||    790.7 |     538.0 |     252.8 |pe.55
```

Using 64 cores on a node creates less available space per MPI rank.

With numactl --preferred=1, after MCDRAM Numanode 1 is full, data is allocated to DRAM Numanode 0.

MCDRAM Allocation Assistance Recap

Will not work
with Intel Compiler

- Cray Tools track requests to memory and evaluate the bandwidth contribution of objects within a program
- Helpful for memory-intensive programs that cannot fit within MCDRAM
- Reduces time investment associated with selectively allocating data into KNL's MCDRAM
- The result is performance portable code
  - CCE memory allocation directives are ignored on X86 processors

Will not work with Intel Compiler

Cray Compiler Optimization Feedback

OpenMP Assistance

MCDRAM Allocation Assistance

Reveal Overview

- Reduce effort associated with adding OpenMP to MPI programs
- Produce performance portable code
- Get insight into optimizations performed by the Cray compiler
- Use as a first step to parallelize loops that will target GPUs
- Track requests to memory and evaluate the bandwidth contribution of objects within a program

Will not work
with Intel Compiler

When to Move to a Hybrid Programming Model

- **When code is network bound**
  - Increased MPI collective and point-to-point wait times
- **When MPI starts leveling off**
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue
- **When contention of shared resources increases**

Approach to Adding Parallelism

1. **Identify key high-level loops**
   - Determine where to add additional levels of parallelism
2. **Perform parallel analysis and scoping**
   - Split loop work among threads
3. **Add OpenMP layer of parallelism**
   - Insert OpenMP directives
4. **Analyze performance for further optimization, specifically vectorization of innermost loops**
   - We want a performance-portable application at the end

Will not work with Intel Compiler

The Problem – How Do I Parallelize This Loop?

- How do I know this is a good loop to parallelize?
- What prevents me from parallelizing this loop?
- Can I get help building a directive?

Will not work
with Intel Compiler

```
subroutine sweepz
…
do j = 1, js
 do i = 1, isz
   radius = zxc(i+mypez*isz)
   theta  = zyc(j+mypey*js)
   do m = 1, npez
    do k = 1, ks
     n = k + ks*(m-1) + 6
     r(n) = recv3(1,j,k,i,m)
     p(n) = recv3(2,j,k,i,m)
     u(n) = recv3(5,j,k,i,m)
     v(n) = recv3(3,j,k,i,m)
     w(n) = recv3(4,j,k,i,m)
     f(n) = recv3(6,j,k,i,m)
    enddo
   enddo
   …
   call ppmlr
   do k = 1, kmax
     n = k + 6
     xa  (n) = zza(k)
     dx  (n) = zdz(k)
     xa0(n) = zza(k)
     dx0(n) = zdz(k)
     e   (n) = p(n)/(r(n)*gamm)+0.5 &
        *(u(n)**2+v(n)**2+w(n)**2)
   enddo
   call ppmlr
…
 enddo
enddo
```

```
subroutine ppmlr

call boundary
call flatten
call paraset(nmin-4, nmax+5, para, dx, xa)

call parabola(nmin-4,nmax+4,para,p,dp,p6,pl,flat)
call parabola(nmin-4,nmax+4, para,r,dr,r6,rl,flat)
call parabola(nmin-4,nmax+4,para,u,du,u6,ul,flat)

call states(pl,ul,rl,p6,u6,r6,dp,du,dr,plft,ulft,&
            rlft,prgh,urgh,rrgh)
call riemann(nmin-3,nmax+4,gam,prgh,urgh,rrgh,&
             plft,ulft,rlft pmid umid)
call evolve(umid, pmid)  ← contains more calls

call remap ← contains more calls

call volume(nmin,nmax,ngeom,radius,xa,dx,dvol)

call remap ← contains more calls

return
end
```

Loop Work Estimates

Gather loop statistics using the Cray performance tools and CCE to determine which loops have the most work

- **Helps identify high-level serial loops to parallelize**
  - Based on runtime analysis, approximates how much work exists within a loop

Will not work
with Intel Compiler

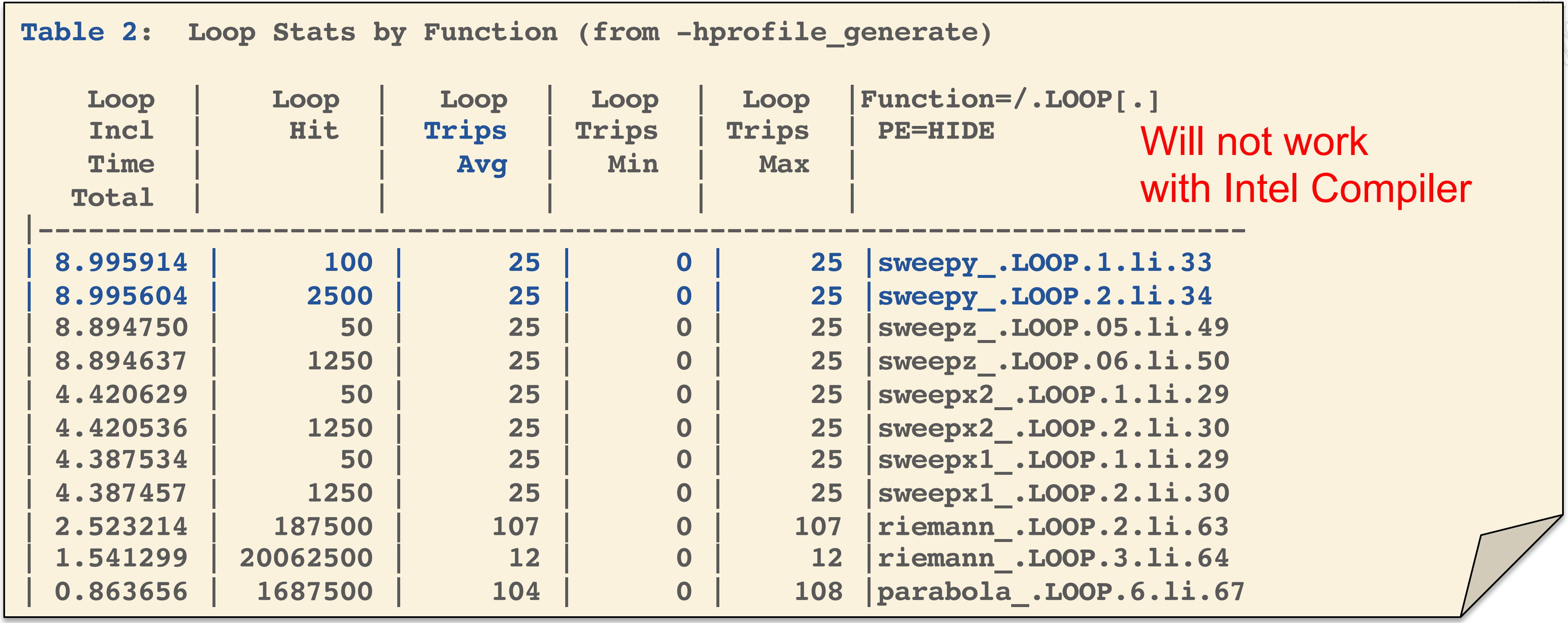
Collect Loop Work Estimates

- Point to Cray compiler
  - `$ module load PrgEnv-cray`

Will not work
with Intel Compiler

- Set up perftools loop work estimates experiment
  - `$ module load perftools-lite-loops`
- Build program (make)
- Run program to get loop work estimates in file with .ap2 suffix

Example Loop Statistics



```
Table 2:  Loop Stats by Function (from -hprofile_generate)

    Loop  |     Loop  |    Loop  |    Loop  |    Loop  |Function=/.LOOP[.]
    Incl  |      Hit  |   Trips  |   Trips  |   Trips  | PE=HIDE
    Time  |           |     Avg  |     Min  |     Max  |
   Total  |           |          |          |          |
|------------------------------------------------------------------------
| 8.995914 |      100 |       25 |        0 |      25 |sweepy_.LOOP.1.li.33
| 8.995604 |     2500 |       25 |        0 |      25 |sweepy_.LOOP.2.li.34
| 8.894750 |       50 |       25 |        0 |      25 |sweepz_.LOOP.05.li.49
| 8.894637 |     1250 |       25 |        0 |      25 |sweepz_.LOOP.06.li.50
| 4.420629 |       50 |       25 |        0 |      25 |sweepx2_.LOOP.1.li.29
| 4.420536 |     1250 |       25 |        0 |      25 |sweepx2_.LOOP.2.li.30
| 4.387534 |       50 |       25 |        0 |      25 |sweepx1_.LOOP.1.li.29
| 4.387457 |     1250 |       25 |        0 |      25 |sweepx1_.LOOP.2.li.30
| 2.523214 |   187500 |      107 |        0 |     107 |riemann_.LOOP.2.li.63
| 1.541299 | 20062500 |       12 |        0 |      12 |riemann_.LOOP.3.li.64
| 0.863656 |  1687500 |      104 |        0 |     108 |parabola_.LOOP.6.li.67
```

Will not work with Intel Compiler

Create Program Library and Launch Reveal

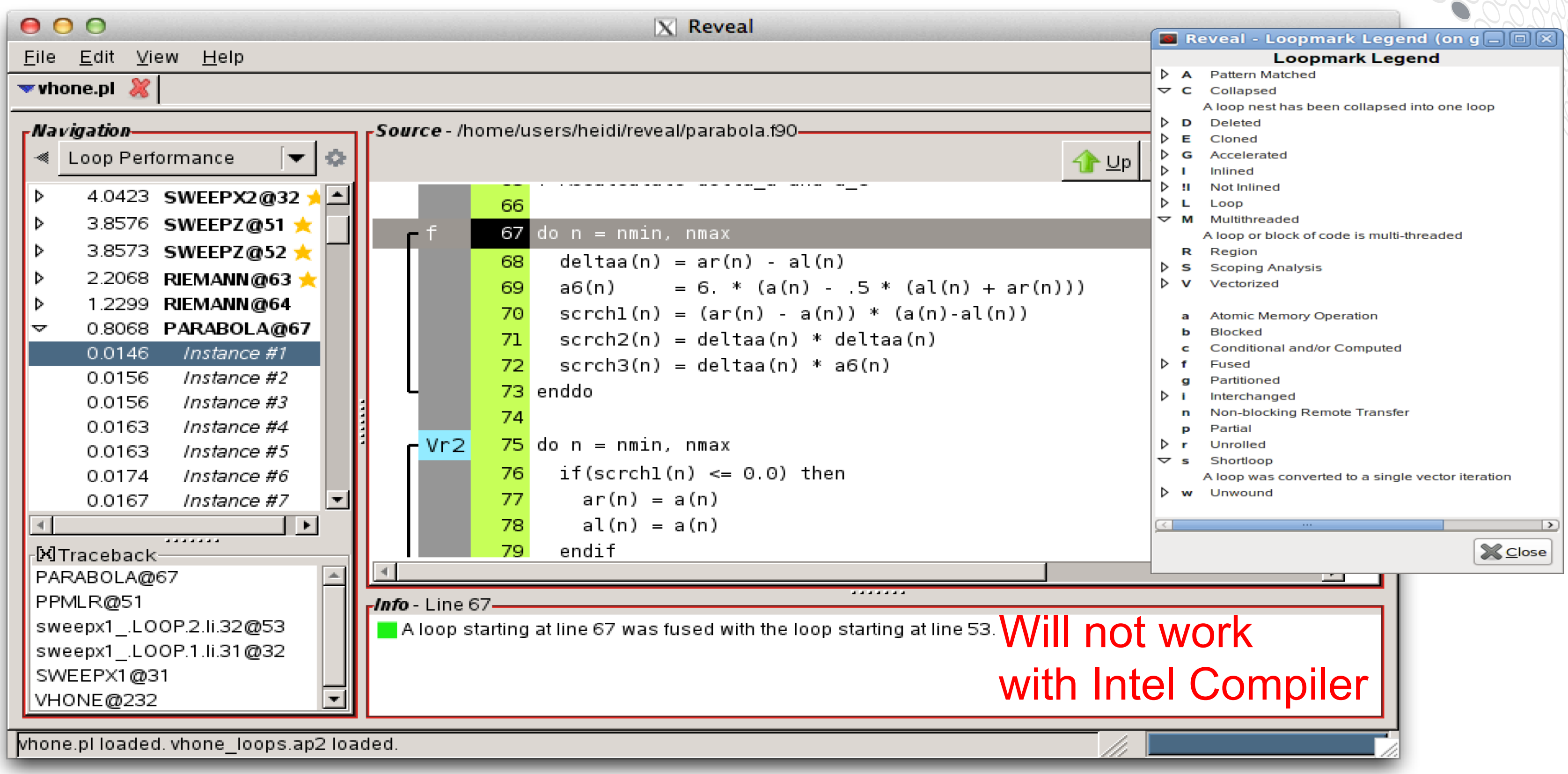
- Disable loop work estimates program instrumentation
  - `$ module unload perftools-lite-loops`

- Rebuild program with CCE's program library
  - Add `-h pl=/full_path/program.pl` to program's Makefile
  - `$ make clean`
  - `$ make`

Will not work with Intel Compiler

- Launch Reveal
  - `$ reveal /full_path/program.pl  loop_estimates.ap2`

View Source and Optimization Information


Reveal
File Edit View Help
vhone.pl
Navigation
Loop Performance
Source - /home/users/heidi/reveal/parabola.f90
Up
Traceback
Reveal - Loopmark Legend (on g
Close

4.0423	SWEEPX2@32 ★
3.8576	SWEEPZ@51 ★
3.8573	SWEEPZ@52 ★
2.2068	RIEMANN@63 ★
1.2299	RIEMANN@64
0.8068	PARABOLA@67
0.0146	*Instance #1*
0.0156	*Instance #2*
0.0156	*Instance #3*
0.0163	*Instance #4*
0.0163	*Instance #5*
0.0174	*Instance #6*
0.0167	*Instance #7*

Traceback

PARABOLA@67
PPMLR@51
sweepx1_.LOOP.2.li.32@53
sweepx1_.LOOP.1.li.31@32
SWEEPX1@31
VHONE@232

```
        66
   f    67 do n = nmin, nmax
        68   deltaa(n) = ar(n) - al(n)
        69   a6(n)     = 6. * (a(n) - .5 * (al(n) + ar(n)))
        70   scrch1(n) = (ar(n) - a(n)) * (a(n)-al(n))
        71   scrch2(n) = deltaa(n) * deltaa(n)
        72   scrch3(n) = deltaa(n) * a6(n)
        73 enddo
        74
   Vr2  75 do n = nmin, nmax
        76   if(scrch1(n) <= 0.0) then
        77     ar(n) = a(n)
        78     al(n) = a(n)
        79   endif
```

Info - Line 67

🟩 A loop starting at line 67 was fused with the loop starting at line 53.

vhone.pl loaded. vhone_loops.ap2 loaded.

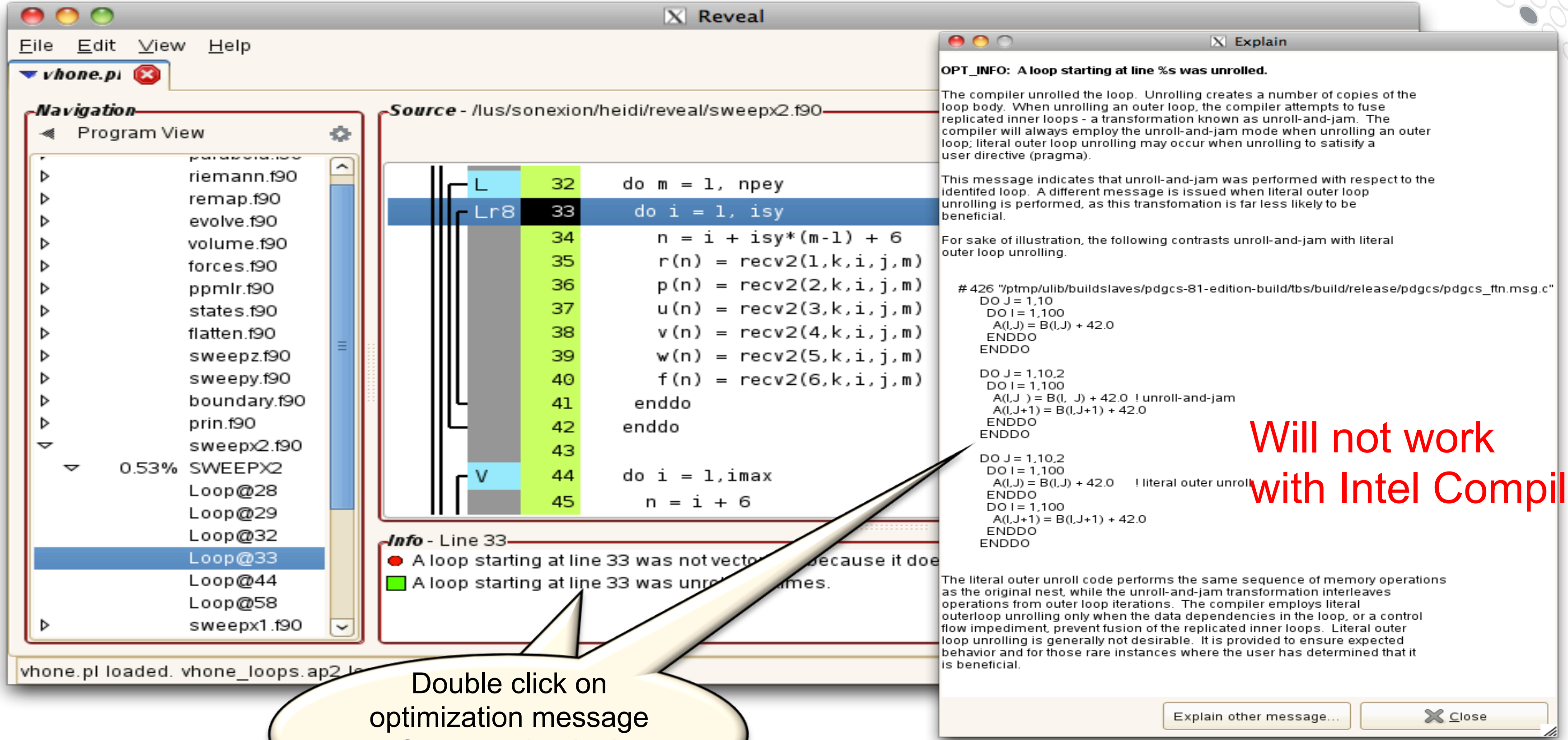
Loopmark Legend

A Pattern Matched
C Collapsed
A loop nest has been collapsed into one loop
D Deleted
E Cloned
G Accelerated
I Inlined
!I Not Inlined
L Loop
M Multithreaded
A loop or block of code is multi-threaded
R Region
S Scoping Analysis
V Vectorized

a Atomic Memory Operation
b Blocked
c Conditional and/or Computed
f Fused
g Partitioned
i Interchanged
n Non-blocking Remote Transfer
p Partial
r Unrolled
s Shortloop
A loop was converted to a single vector iteration
w Unwound

Will not work with Intel Compiler

Access Cray Compiler Message Information



Source - /lus/sonexion/heidi/reveal/sweepx2.f90

```
L    32    do m = 1, npey
Lr8  33      do i = 1, isy
     34        n = i + isy*(m-1) + 6
     35        r(n) = recv2(1,k,i,j,m)
     36        p(n) = recv2(2,k,i,j,m)
     37        u(n) = recv2(3,k,i,j,m)
     38        v(n) = recv2(4,k,i,j,m)
     39        w(n) = recv2(5,k,i,j,m)
     40        f(n) = recv2(6,k,i,j,m)
     41      enddo
     42    enddo
     43
V    44    do i = 1,imax
     45      n = i + 6
```

Info - Line 33

- 🟥 A loop starting at line 33 was not vecto ecause it doe
- 🟩 A loop starting at line 33 was unro mes.

vhone.pl loaded. vhone_loops.ap2 l

OPT_INFO: A loop starting at line %s was unrolled.

The compiler unrolled the loop. Unrolling creates a number of copies of the loop body. When unrolling an outer loop, the compiler attempts to fuse replicated inner loops - a transformation known as unroll-and-jam. The compiler will always employ the unroll-and-jam mode when unrolling an outer loop; literal outer loop unrolling may occur when unrolling to satisify a user directive (pragma).

This message indicates that unroll-and-jam was performed with respect to the identifed loop. A different message is issued when literal outer loop unrolling is performed, as this transfomation is far less likely to be beneficial.

For sake of illustration, the following contrasts unroll-and-jam with literal outer loop unrolling.

```
  # 426 "/ptmp/ulib/buildslaves/pdgcs-81-edition-build/tbs/build/release/pdgcs/pdgcs_ftn.msg.c"
      DO J = 1,10
       DO I = 1,100
        A(I,J) = B(I,J) + 42.0
       ENDDO
      ENDDO

      DO J = 1,10,2
       DO I = 1,100
        A(I,J  ) = B(I,  J) + 42.0  ! unroll-and-jam
        A(I,J+1) = B(I,J+1) + 42.0
       ENDDO
      ENDDO

      DO J = 1,10,2
       DO I = 1,100
        A(I,J) = B(I,J) + 42.0     ! literal outer unroll
       ENDDO
       DO I = 1,100
        A(I,J+1) = B(I,J+1) + 42.0
       ENDDO
      ENDDO
```

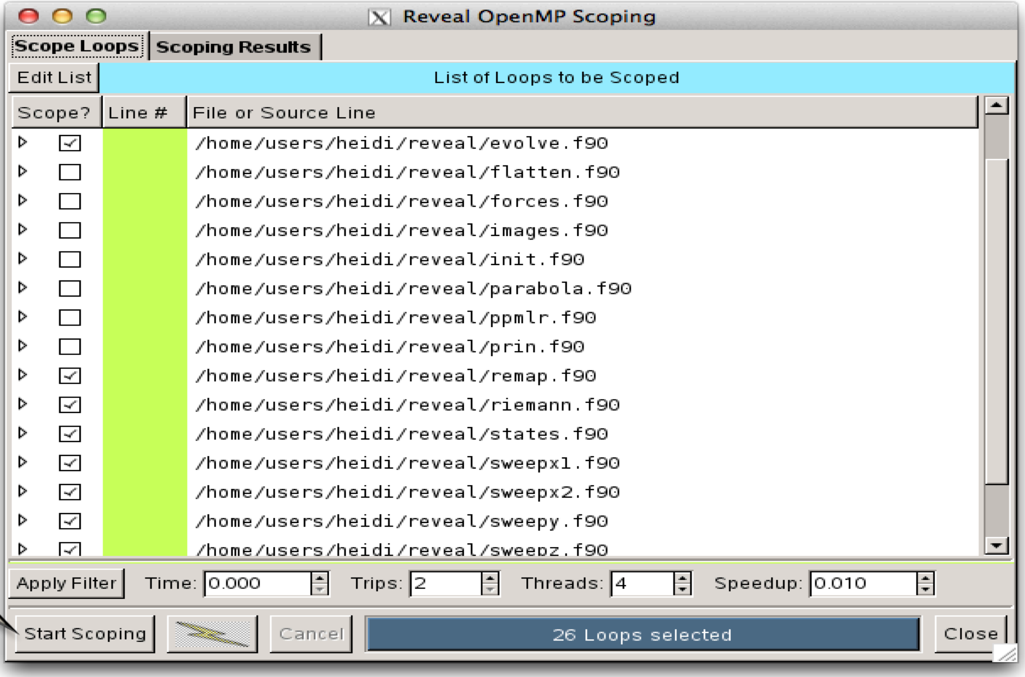
The literal outer unroll code performs the same sequence of memory operations as the original nest, while the unroll-and-jam transformation interleaves operations from outer loop iterations. The compiler employs literal outerloop unrolling only when the data dependencies in the loop, or a control flow impediment, prevent fusion of the replicated inner loops. Literal outer loop unrolling is generally not desirable. It is provided to ensure expected behavior and for those rare instances where the user has determined that it is beneficial.

Will not work with Intel Compiler

Double click on optimization message for more detailed information

Scope Selected Loop(s)

Will not work
with Intel Compiler



Scope?	Line #	File or Source Line
▷ ☑		/home/users/heidi/reveal/evolve.f90
▷ □		/home/users/heidi/reveal/flatten.f90
▷ □		/home/users/heidi/reveal/forces.f90
▷ □		/home/users/heidi/reveal/images.f90
▷ □		/home/users/heidi/reveal/init.f90
▷ □		/home/users/heidi/reveal/parabola.f90
▷ □		/home/users/heidi/reveal/ppmlr.f90
▷ □		/home/users/heidi/reveal/prin.f90
▷ ☑		/home/users/heidi/reveal/remap.f90
▷ ☑		/home/users/heidi/reveal/riemann.f90
▷ ☑		/home/users/heidi/reveal/states.f90
▷ ☑		/home/users/heidi/reveal/sweepx1.f90
▷ ☑		/home/users/heidi/reveal/sweepx2.f90
▷ ☑		/home/users/heidi/reveal/sweepy.f90
▷ ☑		/home/users/heidi/reveal/sweepz.f90

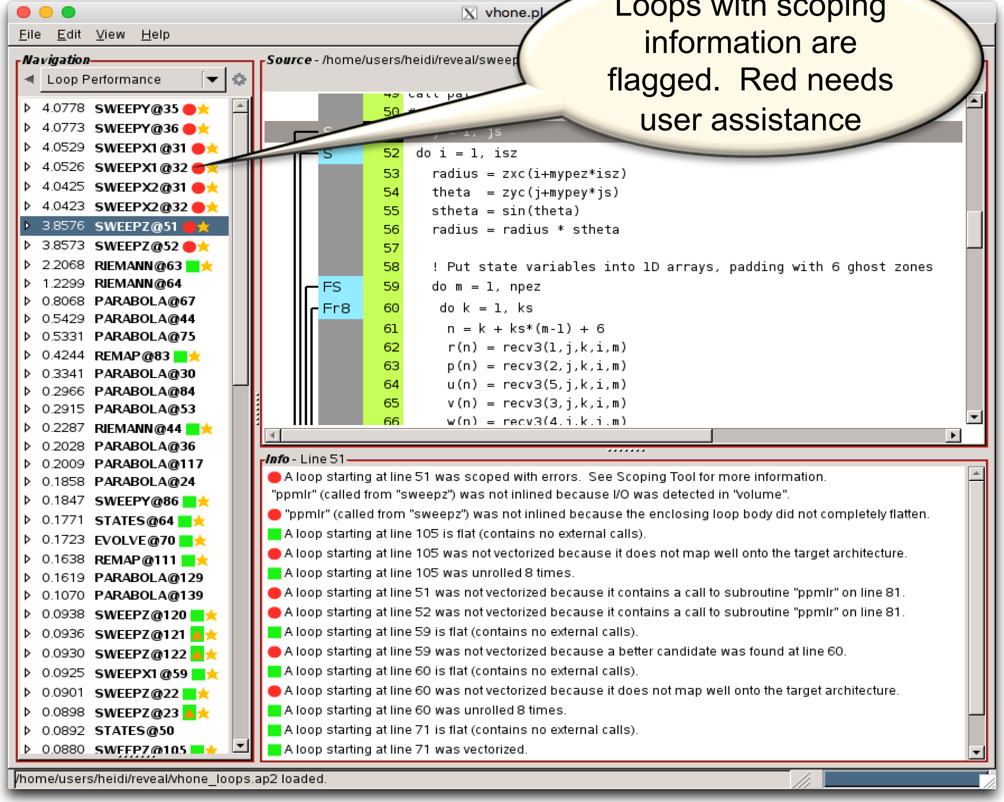
Time: 0.000 Trips: 2 Threads: 4 Speedup: 0.010

26 Loops selected

- Trigger dependence analysis
- scope loops above given threshold

Review Scoping Results

Will not work
with Intel Compiler



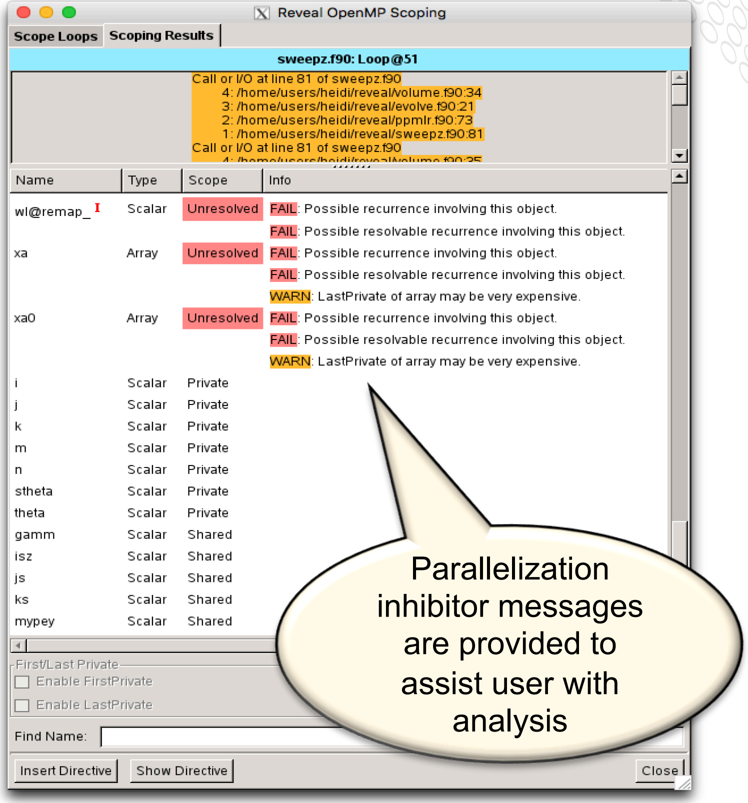
4.0778	SWEEPY@35	● ★
4.0773	SWEEPY@36	● ★
4.0529	SWEEPX1@31	● ★
4.0526	SWEEPX1@32	● ★
4.0425	SWEEPX2@31	● ★
4.0423	SWEEPX2@32	● ★
3.8576	SWEEPZ@51	● ★
3.8573	SWEEPZ@52	● ★
2.2068	RIEMANN@63	🟩 ★
1.2299	RIEMANN@64	
0.8068	PARABOLA@67	
0.5429	PARABOLA@44	
0.5331	PARABOLA@75	
0.4244	REMAP@83	🟩 ★
0.3341	PARABOLA@30	
0.2966	PARABOLA@84	
0.2915	PARABOLA@53	
0.2287	RIEMANN@44	🟩 ★
0.2028	PARABOLA@36	
0.2009	PARABOLA@117	
0.1858	PARABOLA@24	
0.1847	SWEEPY@86	🟩 ★
0.1771	STATES@64	🟩 ★
0.1723	EVOLVE@70	🟩 ★
0.1638	REMAP@111	🟩 ★
0.1619	PARABOLA@129	
0.1070	PARABOLA@139	
0.0938	SWEEPZ@120	🟩 ★
0.0936	SWEEPZ@121	▲ ★
0.0930	SWEEPZ@122	▲ ★
0.0925	SWEEPX1@59	🟩 ★
0.0901	SWEEPZ@22	🟩 ★
0.0898	SWEEPZ@23	▲ ★
0.0892	STATES@50	
0.0880	SWEEPZ@105	🟩 ★

```
   S    52  do i = 1, isz
        53    radius = zxc(i+mypez*isz)
        54    theta  = zyc(j+mypey*js)
        55    stheta = sin(theta)
        56    radius = radius * stheta
        57
        58    ! Put state variables into 1D arrays, padding with 6 ghost zones
   FS   59    do m = 1, npez
   Fr8  60     do k = 1, ks
        61      n = k + ks*(m-1) + 6
        62      r(n) = recv3(1,j,k,i,m)
        63      p(n) = recv3(2,j,k,i,m)
        64      u(n) = recv3(5,j,k,i,m)
        65      v(n) = recv3(3,j,k,i,m)
        66      w(n) = recv3(4,j,k,i,m)
```

- ● A loop starting at line 51 was scoped with errors. See Scoping Tool for more information.
  "ppmlr" (called from "sweepz") was not inlined because I/O was detected in "volume".
- ● "ppmlr" (called from "sweepz") was not inlined because the enclosing loop body did not completely flatten.
- 🟩 A loop starting at line 105 is flat (contains no external calls).
- ● A loop starting at line 105 was not vectorized because it does not map well onto the target architecture.
- 🟩 A loop starting at line 105 was unrolled 8 times.
- ● A loop starting at line 51 was not vectorized because it contains a call to subroutine "ppmlr" on line 81.
- ● A loop starting at line 52 was not vectorized because it contains a call to subroutine "ppmlr" on line 81.
- 🟩 A loop starting at line 59 is flat (contains no external calls).
- ● A loop starting at line 59 was not vectorized because a better candidate was found at line 60.
- 🟩 A loop starting at line 60 is flat (contains no external calls).
- ● A loop starting at line 60 was not vectorized because it does not map well onto the target architecture.
- 🟩 A loop starting at line 60 was unrolled 8 times.
- 🟩 A loop starting at line 71 is flat (contains no external calls).
- 🟩 A loop starting at line 71 was vectorized.

/home/users/heidi/reveal/vhone_loops.ap2 loaded.

Loops with scoping information are flagged. Red needs user assistance



sweepz.f90: Loop@51

```
Call or I/O at line 81 of sweepz.f90
    4: /home/users/heidi/reveal/volume.f90:34
    3: /home/users/heidi/reveal/evolve.f90:21
    2: /home/users/heidi/reveal/ppmlr.f90:73
    1: /home/users/heidi/reveal/sweepz.f90:81
Call or I/O at line 81 of sweepz.f90
```

Name	Type	Scope	Info
wl@remap_	Scalar	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object.
xa	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
xa0	Array	Unresolved	FAIL: Possible recurrence involving this object. FAIL: Possible resolvable recurrence involving this object. WARN: LastPrivate of array may be very expensive.
i	Scalar	Private	
j	Scalar	Private	
k	Scalar	Private	
m	Scalar	Private	
n	Scalar	Private	
stheta	Scalar	Private	
theta	Scalar	Private	
gamm	Scalar	Shared	
isz	Scalar	Shared	
js	Scalar	Shared	
ks	Scalar	Shared	
mypey	Scalar	Shared	

First/Last Private

☐ Enable FirstPrivate

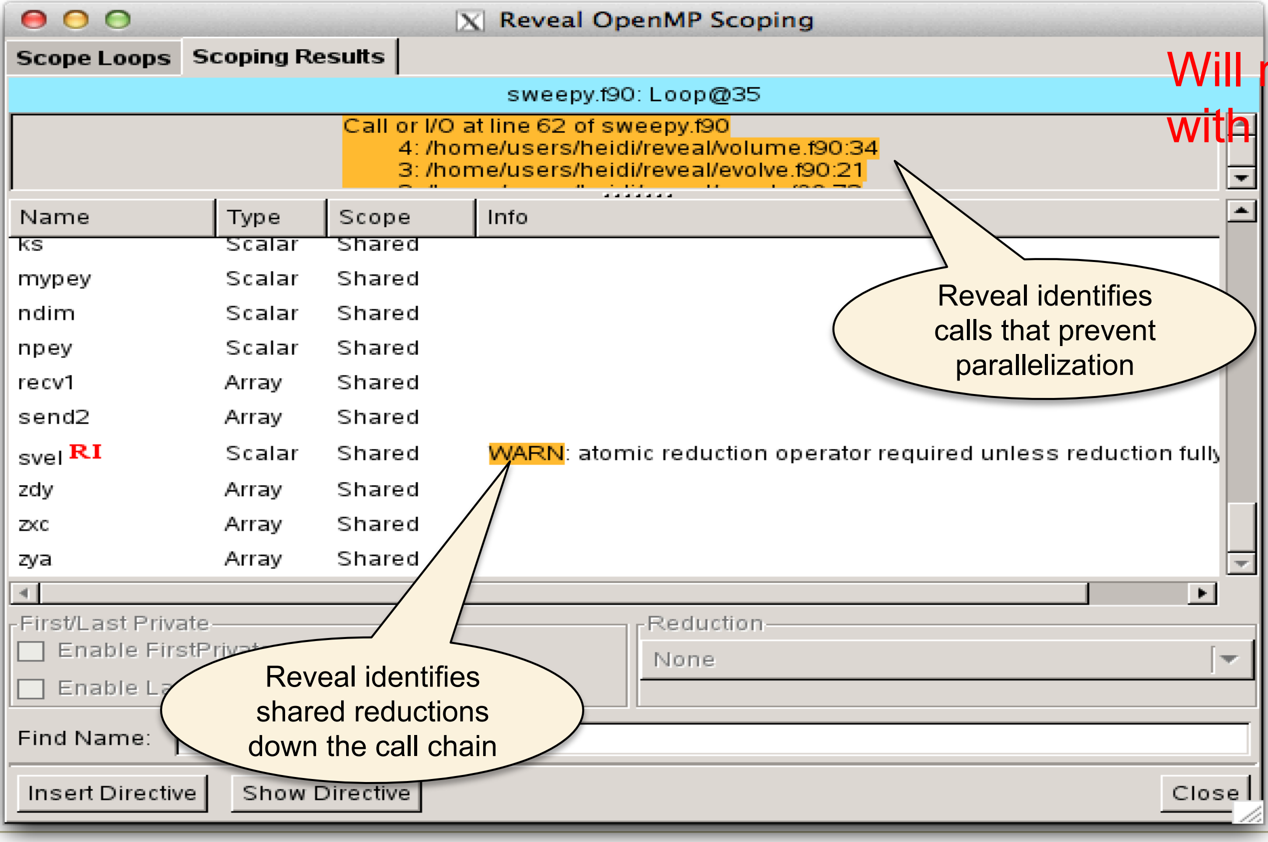
☐ Enable LastPrivate

Find Name:

Parallelization inhibitor messages are provided to assist user with analysis

Review Scoping Results (continued)

Will not work with Intel Compiler



sweepy.f90: Loop@35

```
Call or I/O at line 62 of sweepy.f90
    4: /home/users/heidi/reveal/volume.f90:34
    3: /home/users/heidi/reveal/evolve.f90:21
```

Name	Type	Scope	Info
ks	Scalar	Shared	
mypey	Scalar	Shared	
ndim	Scalar	Shared	
npey	Scalar	Shared	
recv1	Array	Shared	
send2	Array	Shared	
svel RI	Scalar	Shared	WARN: atomic reduction operator required unless reduction fully
zdy	Array	Shared	
zxc	Array	Shared	
zya	Array	Shared	

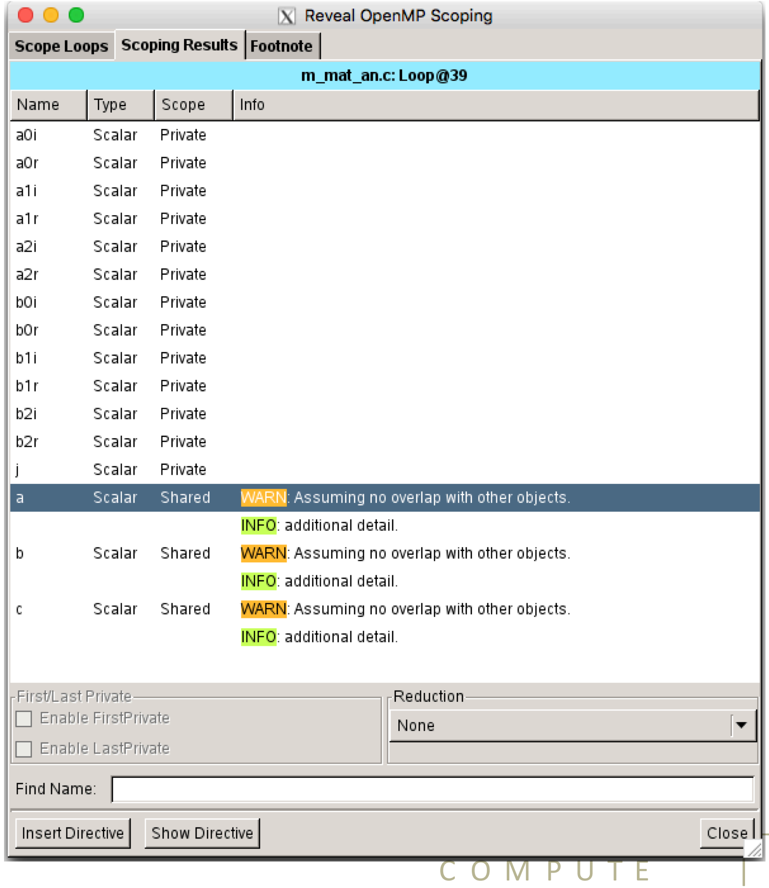
First/Last Private
□ Enable FirstPrivat
□ Enable La

Reduction: None

Reveal identifies calls that prevent parallelization

Reveal identifies shared reductions down the call chain

Review Scoping Results (continued)



m_mat_an.c: Loop@39

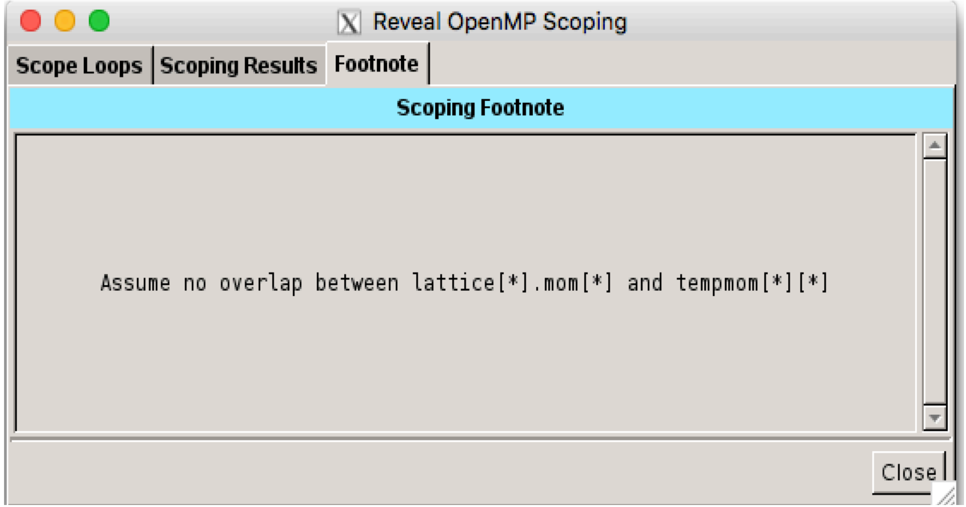
Name	Type	Scope	Info
a0i	Scalar	Private	
a0r	Scalar	Private	
a1i	Scalar	Private	
a1r	Scalar	Private	
a2i	Scalar	Private	
a2r	Scalar	Private	
b0i	Scalar	Private	
b0r	Scalar	Private	
b1i	Scalar	Private	
b1r	Scalar	Private	
b2i	Scalar	Private	
b2r	Scalar	Private	
j	Scalar	Private	
a	Scalar	Shared	WARN: Assuming no overlap with other objects. INFO: additional detail.
b	Scalar	Shared	WARN: Assuming no overlap with other objects. INFO: additional detail.
c	Scalar	Shared	WARN: Assuming no overlap with other objects. INFO: additional detail.

First/Last Private
☐ Enable FirstPrivate
☐ Enable LastPrivate

Reduction: None

Find Name:

Will not work
with Intel Compiler



Scoping Footnote

```
Assume no overlap between lattice[*].mom[*] and tempmom[*][*]
```

Generate OpenMP Directives

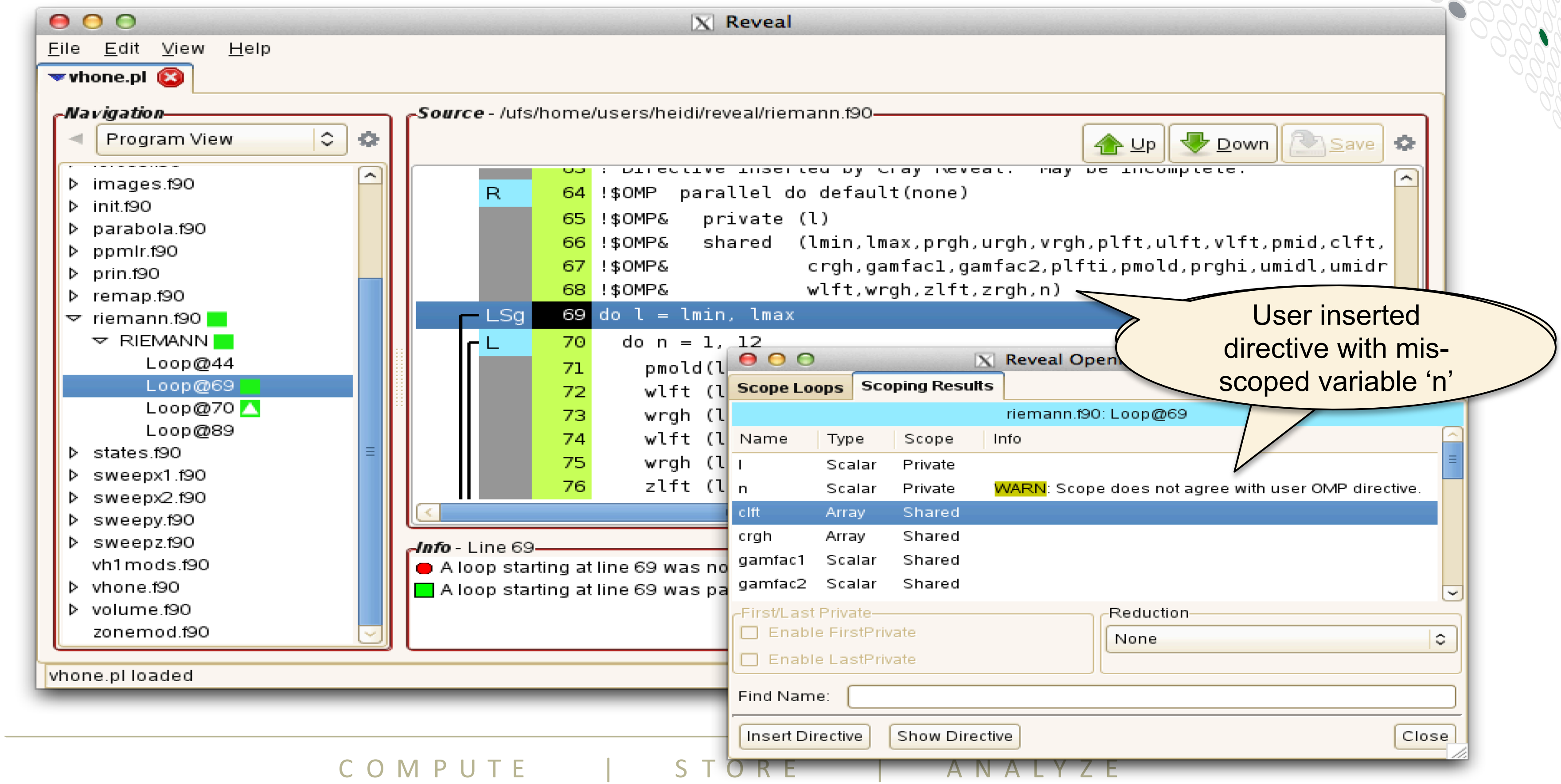
```
! Directive inserted by Cray Reveal.  May be incomplete.
!$OMP  parallel do default(none)                                            &
!$OMP&   unresolved (dvol,dx,dx0,e,f,flat,p,para,q,r,radius,svel,u,v,w,     &
!$OMP&          xa,xa0)                                                     &
!$OMP&   private (i,j,k,m,n,$$_n,delp2,delp1,shock,temp2,old_flat,          &
!$OMP&          onemfl,hdt,sinxf0,gamfac1,gamfac2,dtheta,deltx,fractn,      &
!$OMP&          ekin)                                                       &
!$OMP&   shared  (gamm,isy,js,ks,mypey,ndim,ngeomy,nlefty,npey,nrighty, &
!$OMP&          recv1,send2,zdy,zxc,zya)
do k = 1, ks
 do i = 1, isy
   radius = zxc(i+mypey*isy)

   ! Put state variables into 1D arrays, padding with 6 ghost zones
   do m = 1, npey
    do j = 1, js
     n = j + js*(m-1) + 6
     r(n) = recv1(1,k,j,i,m)
     p(n) = recv1(2,k,j,i,m)
     u(n) = recv1(4,k,j,i,m)
     v(n) = recv1(5,k,j,i,m)
     w(n) = recv1(3,k,j,i,m)
     f(n) = recv1(6,k,j,i,m)
    enddo
   enddo

   do j = 1, jmax
     n = j + 6
```

Will not work
with Intel Compiler

Reveal generates OpenMP directive with illegal clause marking variables that need addressing

Validate User Inserted Directives

Will not work with Intel Compiler



```
    63 ! Directive inserted by Cray Reveal.  May be incomplete.
R   64 !$OMP  parallel do default(none)
    65 !$OMP&   private (l)
    66 !$OMP&   shared  (lmin,lmax,prgh,urgh,vrgh,plft,ulft,vlft,pmid,clft,
    67 !$OMP&              crgh,gamfac1,gamfac2,plfti,pmold,prghi,umidl,umidr
    68 !$OMP&              wlft,wrgh,zlft,zrgh,n)
LSg 69 do l = lmin, lmax
L   70   do n = 1, 12
    71     pmold(l
    72     wlft (l
    73     wrgh (l
    74     wlft (l
    75     wrgh (l
    76     zlft (l
```

Info - Line 69

🟥 A loop starting at line 69 was no

🟩 A loop starting at line 69 was pa

riemann.f90: Loop@69

Name	Type	Scope	Info
l	Scalar	Private	
n	Scalar	Private	WARN: Scope does not agree with user OMP directive.
clft	Array	Shared	
crgh	Array	Shared	
gamfac1	Scalar	Shared	
gamfac2	Scalar	Shared	

☐ Enable FirstPrivate

☐ Enable LastPrivate

Reduction: None

Find Name:

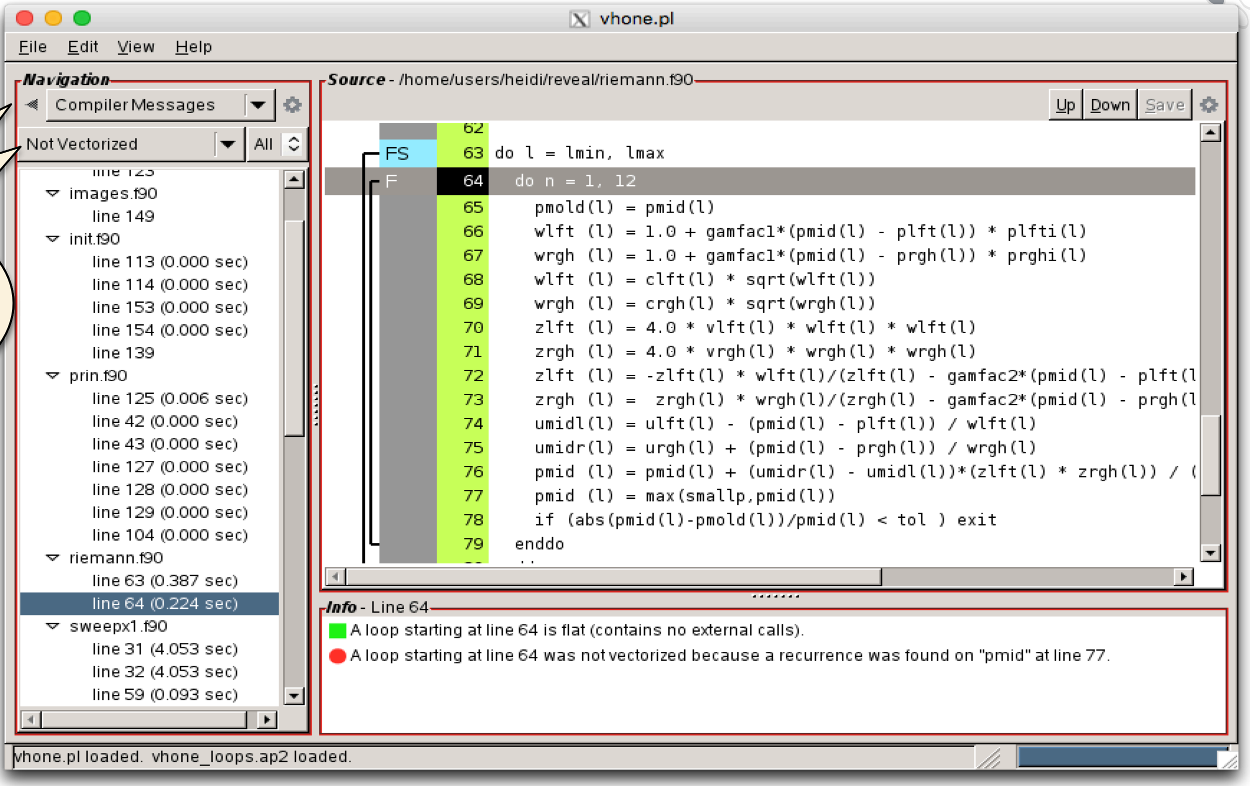
vhone.pl loaded

User inserted directive with mis-scoped variable ‘n’

Look For Vectorization Opportunities

Will not work with Intel Compiler

Choose "Compiler Messages" view to access message filtering, then select desired type of message



- line 123
- images.f90
  - line 149
- init.f90
  - line 113 (0.000 sec)
  - line 114 (0.000 sec)
  - line 153 (0.000 sec)
  - line 154 (0.000 sec)
  - line 139
- prin.f90
  - line 125 (0.006 sec)
  - line 42 (0.000 sec)
  - line 43 (0.000 sec)
  - line 127 (0.000 sec)
  - line 128 (0.000 sec)
  - line 129 (0.000 sec)
  - line 104 (0.000 sec)
- riemann.f90
  - line 63 (0.387 sec)
  - line 64 (0.224 sec)
- sweepx1.f90
  - line 31 (4.053 sec)
  - line 32 (4.053 sec)
  - line 59 (0.093 sec)

```
      62
FS    63 do l = lmin, lmax
F     64   do n = 1, 12
      65     pmold(l) = pmid(l)
      66     wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
      67     wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
      68     wlft (l) = clft(l) * sqrt(wlft(l))
      69     wrgh (l) = crgh(l) * sqrt(wrgh(l))
      70     zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
      71     zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
      72     zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l
      73     zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l
      74     umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
      75     umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
      76     pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (
      77     pmid (l) = max(smallp,pmid(l))
      78     if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
      79   enddo
```

Info - Line 64

🟩 A loop starting at line 64 is flat (contains no external calls).

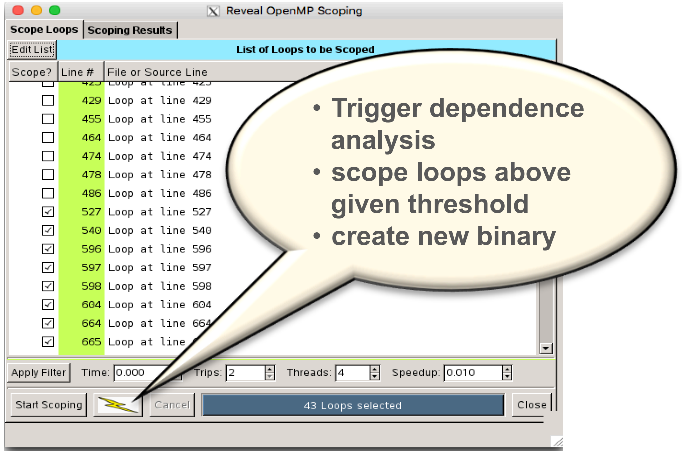
● A loop starting at line 64 was not vectorized because a recurrence was found on "pmid" at line 77.

vhone.pl loaded. vhone_loops.ap2 loaded.

Reveal Auto-Parallelization

Will not work
with Intel Compiler

- Build an *experimental binary* that includes **automatic runtime-assisted parallelization**
- **No source code changes required** to see if high level loops that contain calls can be automatically parallelized
- Result includes **parallelization** of serial loops **via traditional OpenMP** as well as **more extensive loop optimizations**
- User Workflow:
  1. Obtain **loop work estimates** using CCE and perftools-lite-loops
  2. Use Reveal and CCE's program library to **parallelize loops and create experimental binary**
  3. Run experimental binary and compare performance against baseline
  4. If auto-parallelization is successful, **use Reveal to insert parallel directives into source**



Scope?	Line #	File or Source Line
□	425	Loop at line 425
□	429	Loop at line 429
□	455	Loop at line 455
□	464	Loop at line 464
□	474	Loop at line 474
□	478	Loop at line 478
□	486	Loop at line 486
☑	527	Loop at line 527
☑	540	Loop at line 540
☑	596	Loop at line 596
☑	597	Loop at line 597
☑	598	Loop at line 598
☑	604	Loop at line 604
☑	664	Loop at line 664
☑	665	Loop at line

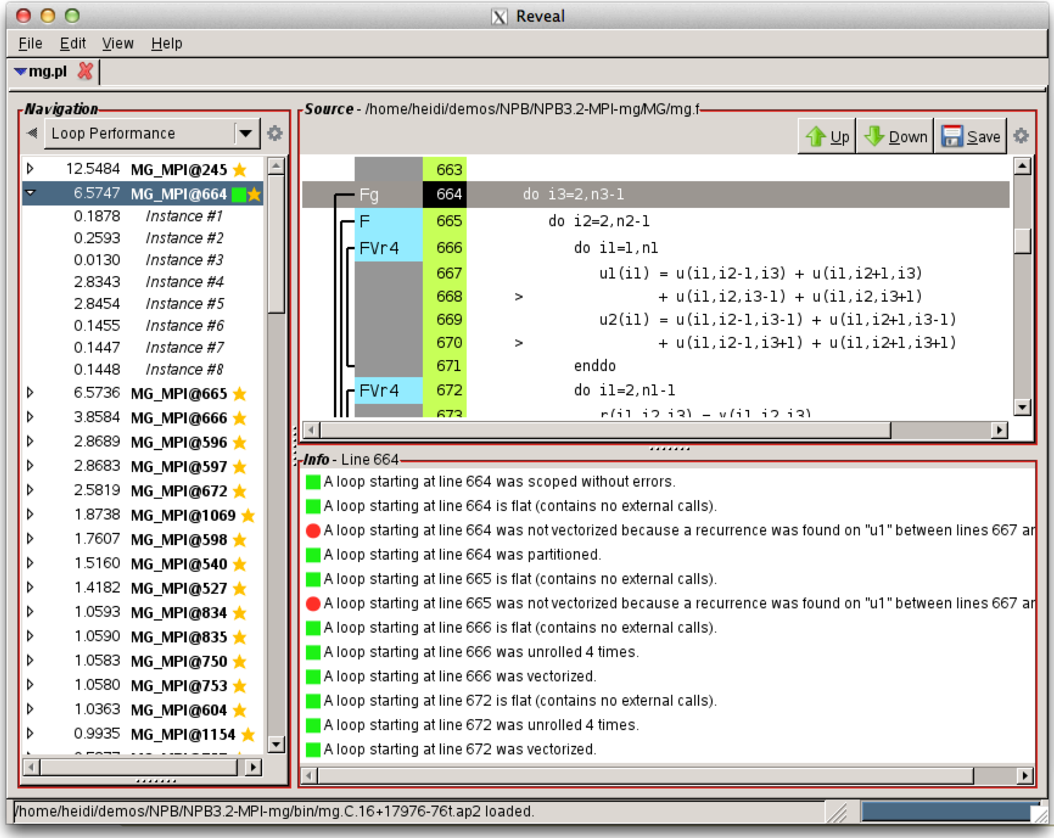
Time: 0.000 Trips: 2 Threads: 4 Speedup: 0.010

43 Loops selected

- Trigger dependence analysis
- scope loops above given threshold
- create new binary

Reveal Analysis Feedback

Will not work
with Intel Compiler

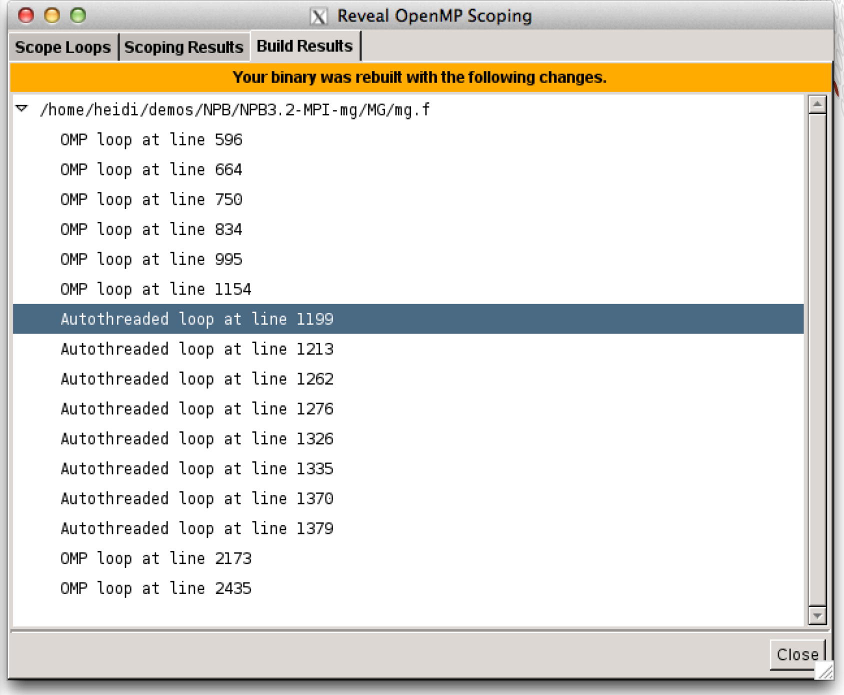


12.5484	MG_MPI@245 ★
6.5747	MG_MPI@664 🟩★
0.1878	*Instance #1*
0.2593	*Instance #2*
0.0130	*Instance #3*
2.8343	*Instance #4*
2.8454	*Instance #5*
0.1455	*Instance #6*
0.1447	*Instance #7*
0.1448	*Instance #8*
6.5736	MG_MPI@665 ★
3.8584	MG_MPI@666 ★
2.8689	MG_MPI@596 ★
2.8683	MG_MPI@597 ★
2.5819	MG_MPI@672 ★
1.8738	MG_MPI@1069 ★
1.7607	MG_MPI@598 ★
1.5160	MG_MPI@540 ★
1.4182	MG_MPI@527 ★
1.0593	MG_MPI@834 ★
1.0590	MG_MPI@835 ★
1.0583	MG_MPI@750 ★
1.0580	MG_MPI@753 ★
1.0363	MG_MPI@604 ★
0.9935	MG_MPI@1154 ★

```
        663
Fg      664        do i3=2,n3-1
F       665           do i2=2,n2-1
FVr4    666              do i1=1,n1
        667                 u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
        668     >                  + u(i1,i2,i3-1) + u(i1,i2,i3+1)
        669                 u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
        670     >                  + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
        671              enddo
FVr4    672              do i1=2,n1-1
        673                 r(i1,i2,i3) = v(i1,i2,i3)
```

🟩 A loop starting at line 664 was scoped without errors.
🟩 A loop starting at line 664 is flat (contains no external calls).
🔴 A loop starting at line 664 was not vectorized because a recurrence was found on "u1" between lines 667 ar
🟩 A loop starting at line 664 was partitioned.
🟩 A loop starting at line 665 is flat (contains no external calls).
🔴 A loop starting at line 665 was not vectorized because a recurrence was found on "u1" between lines 667 ar
🟩 A loop starting at line 666 is flat (contains no external calls).
🟩 A loop starting at line 666 was unrolled 4 times.
🟩 A loop starting at line 666 was vectorized.
🟩 A loop starting at line 672 is flat (contains no external calls).
🟩 A loop starting at line 672 was unrolled 4 times.
🟩 A loop starting at line 672 was vectorized.

/home/heidi/demos/NPB/NPB3.2-MPI-mg/bin/mg.C.16+17976-76t.ap2 loaded.



Your binary was rebuilt with the following changes.

```
/home/heidi/demos/NPB/NPB3.2-MPI-mg/MG/mg.f
  OMP loop at line 596
  OMP loop at line 664
  OMP loop at line 750
  OMP loop at line 834
  OMP loop at line 995
  OMP loop at line 1154
  Autothreaded loop at line 1199
  Autothreaded loop at line 1213
  Autothreaded loop at line 1262
  Autothreaded loop at line 1276
  Autothreaded loop at line 1326
  Autothreaded loop at line 1335
  Autothreaded loop at line 1370
  Autothreaded loop at line 1379
  OMP loop at line 2173
  OMP loop at line 2435
```

Reveal Auto-Parallelization Recap

- **Minimal user time investment includes time to set up and run optimization experiment**
  - Collect loop work estimates
  - Build program library
  - Click button in Reveal
  - Run experimental binary and compare against original program

Will not work
with Intel Compiler

- **Even if experiment does not yield a performance improvement, Reveal will provide insight into parallelization issues**

- **Target: when a pure MPI solution does not scale**

- **Use on X86 hardware (AMD Interlagos, Intel Broadwell, etc.)**

Optimal Programming for the KNL

- **First and foremost, make sure you are taking advantage of available memory bandwidth**
  - If you are memory bandwidth limited, vectorization and threading will not help
  - If on the other hand, if you take steps to improve memory bandwidth utilization then vectorization and threading will give some improvement
- **If you do not vectorize or cannot not vectorize your application, KNL will be of little value**
- **Bottom line – you must take advantage of MCDRAM and then vectorize**
- **Utilization of available cores then the final important step for taking advantage of the KNL**

Taking advantage of available Bandwidth

- **Need less than 16 Gbytes?   You got it made**
- **Need more than 16 Gbytes**
  - MUST look at cache utilization.
    - KNL has small amount of lower level cache, hard to block for Lower level cache; however, blocking for MCDRAM is important and doable
  - Striding and indirect address needs to be examined
    - Can it be eliminated?
    - Can the working set be blocked to fit into MCDRAM used as Cache?
  - Eliminate unnecessary stores

Vectorization

- Vectorize
- Check Alignment

Explanation of tests

- These are small tests which is not really good for timing on KNL since the cache is always flushed prior to the test. This means that some are latency dominated, so DDR may be faster than MCDRAM.
- Trying to illustrate what the compilers, CCE and Intel do with different types of loops. Performance variations are largely dependent upon cache reuse as much as vectorization

Impact of size of stride

ORGINAL

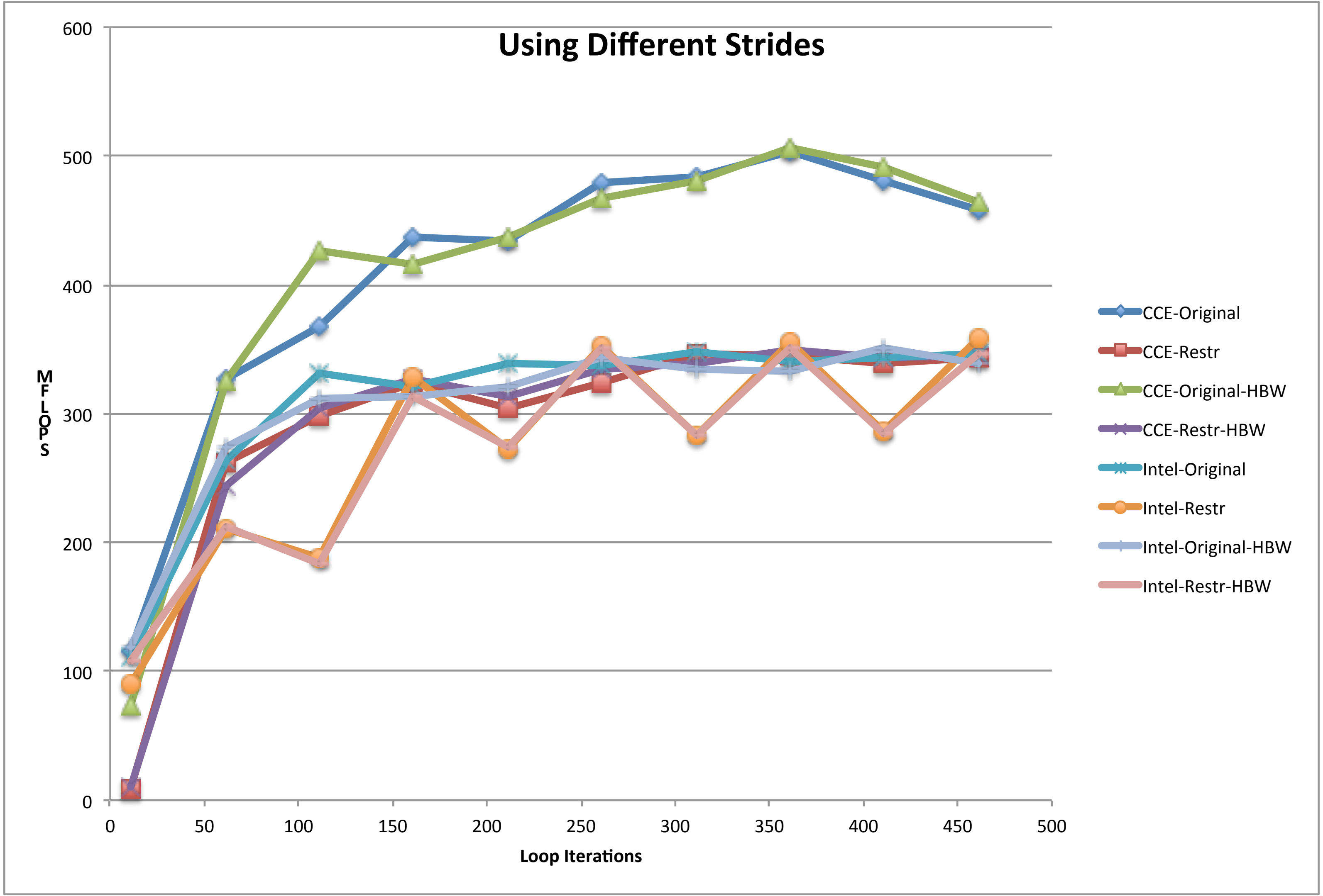
```
48.    1        C     DIMENSION A(128,N)
49.    1
50.    1 Vr2--<       DO 41080  I = 1,N
51.    1 Vr2           A( 1,I) = C1*A(13,I) + C2* A(12,I) + C3*A(11,I) +
52.    1 Vr2        *           C4*A(10,I) + C5* A( 9,I) + C6*A( 8,I) +
53.    1 Vr2        *           C7*A( 7,I) + C0*(A( 5,I) + A( 6,I) )  + A( 3,I)
54.    1 Vr2--> 41080 CONTINUE
```

RESTRUCTURED

```
75.    1        C     DIMENSION B(13,N)
76.    1
77.    1 Vr2--<       DO 41081 I = 1,N
78.    1 Vr2           B( 1,I) = C1*B(13,I) + C2* B(12,I) + C3*B(11,I) +
79.    1 Vr2        *           C4*B(10,I) + C5* B( 9,I) + C6*B( 8,I) +
80.    1 Vr2        *           C7*B( 7,I) + C0*(B( 5,I) + B( 6,I) )  + B( 3,I)
81.    1 Vr2--> 41081 CONTINUE
```

Impact of size of stride
Intel -qopt-report-phase=all

```
LOOP BEGIN at lp41080.f(50,10)
   remark #15300: LOOP WAS VECTORIZED
 LOOP END

 LOOP BEGIN at lp41080.f(50,10)
 <Remainder loop for vectorization>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
 LOOP END

LOOP BEGIN at lp41080.f(77,10)
   remark #15300: LOOP WAS VECTORIZED
 LOOP END

 LOOP BEGIN at lp41080.f(77,10)
 <Remainder loop for vectorization>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
 LOOP END
```


Using Different Strides
MFLOPS
600
500
400
300
200
100
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iterations
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Discussion – Impact of Stride

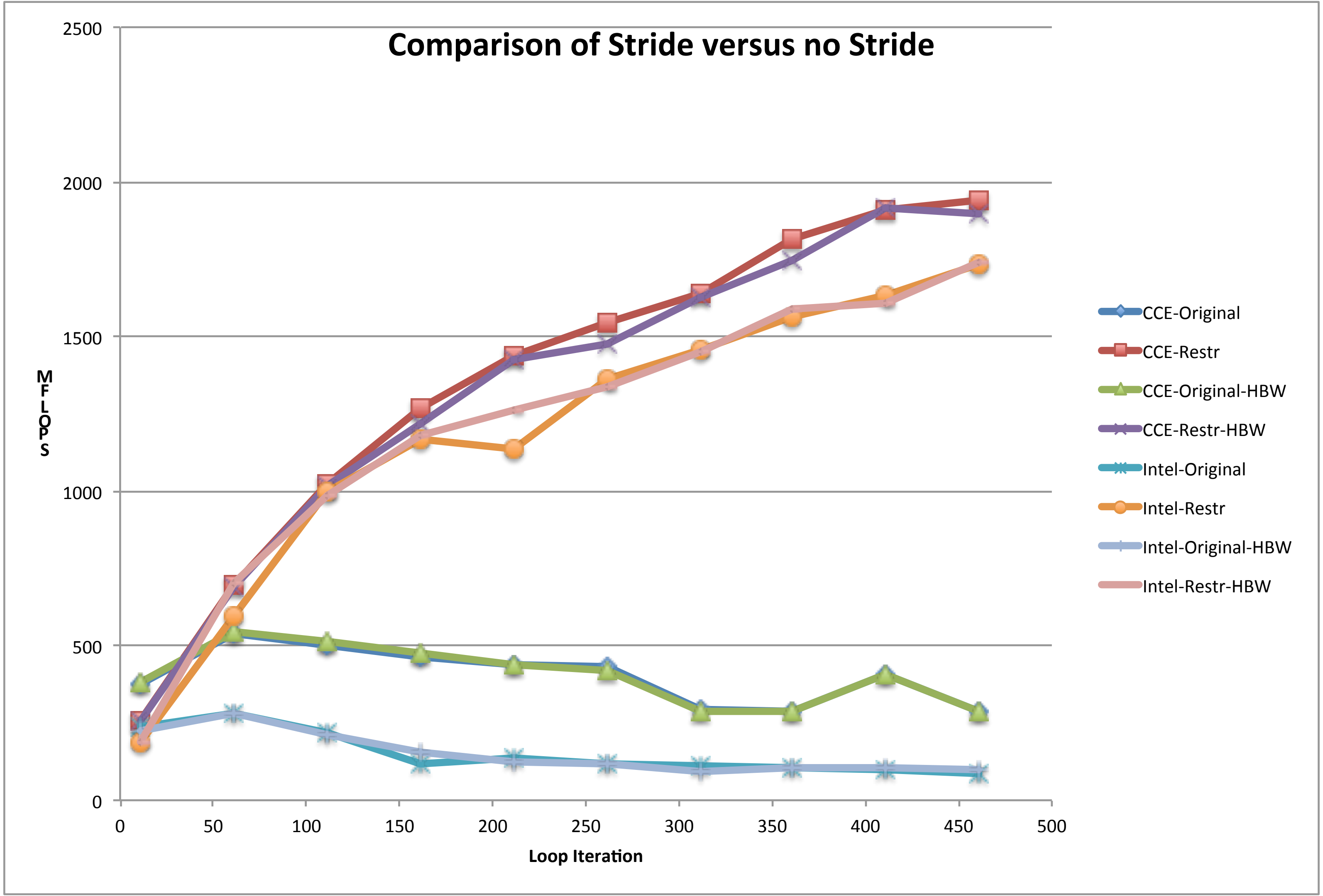
- **My Guess**
  - With the larger stride, CCE used scalar instructions to fetch elements
  - With the smaller stride, CCE and Intel used gather/scatter
- **Need to confirm with Compiler people**

Impact of size of stride

```
 59.     1                 C      THE ORIGINAL
 60.     1
 61.   + 1 2-------------<       DO 41090 K = KA, KE, -1
 62.     1 2 iVp---------<         DO 41090 J = JA, JE
 63.   + 1 2 iVp i-------<           DO 41090 I = IA, IE
 64.     1 2 iVp i                     A(K,L,I,J) = A(K,L,I,J) - B(J,1,i,k)*A(K+1,L,I,1)
 65.     1 2 iVp i                 *     - B(J,2,i,k)*A(K+1,L,I,2) - B(J,3,i,k)*A(K+1,L,I,3)
 66.     1 2 iVp i                 *     - B(J,4,i,k)*A(K+1,L,I,4) - B(J,5,i,k)*A(K+1,L,I,5)
 67.     1 2 iVp i----->>> 41090 CONTINUE

 96.     1                 C      THE RESTRUCTURED
 97.     1
 98.   + 1 2-------------<       DO 41091 K = KA, KE, -1
 99.   + 1 2 3-----------<         DO 41091 J = JA, JE
100.     1 2 3 Vr2-------<           DO 41091 I = IA, IE
101.     1 2 3 Vr2                     AA(I,K,L,J) = AA(I,K,L,J) - BB(I,J,1,K)*AA(I,K+1,L,1)
102.     1 2 3 Vr2                 *     - BB(I,J,2,K)*AA(I,K+1,L,2) - BB(I,J,3,K)*AA(I,K+1,L,3)
103.     1 2 3 Vr2                 *     - BB(I,J,4,K)*AA(I,K+1,L,4) - BB(I,J,5,K)*AA(I,K+1,L,5)
104.     1 2 3 Vr2----->>> 41091 CONTINUE
```

Stride versus no Stride

```
LOOP BEGIN at lp41090.f(48,10)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at lp41090.f(61,10)
      remark #15542: loop was not vectorized: inner loop was already vectorized

      LOOP BEGIN at lp41090.f(62,12)
         remark #15542: loop was not vectorized: inner loop was already vectorized

         LOOP BEGIN at lp41090.f(63,14)
            remark #15300: LOOP WAS VECTORIZED
         LOOP END

         LOOP BEGIN at lp41090.f(63,14)
         <Remainder loop for vectorization>
            remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient.
                                                 Use vector always directive or -vec-threshold0 to override
         LOOP END
      LOOP END
   LOOP END
```


Comparison of Stride versus no Stride
MFLOPS
Loop Iteration
0
500
1000
1500
2000
2500
0
50
100
150
200
250
300
350
400
450
500
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Impact of size of stride

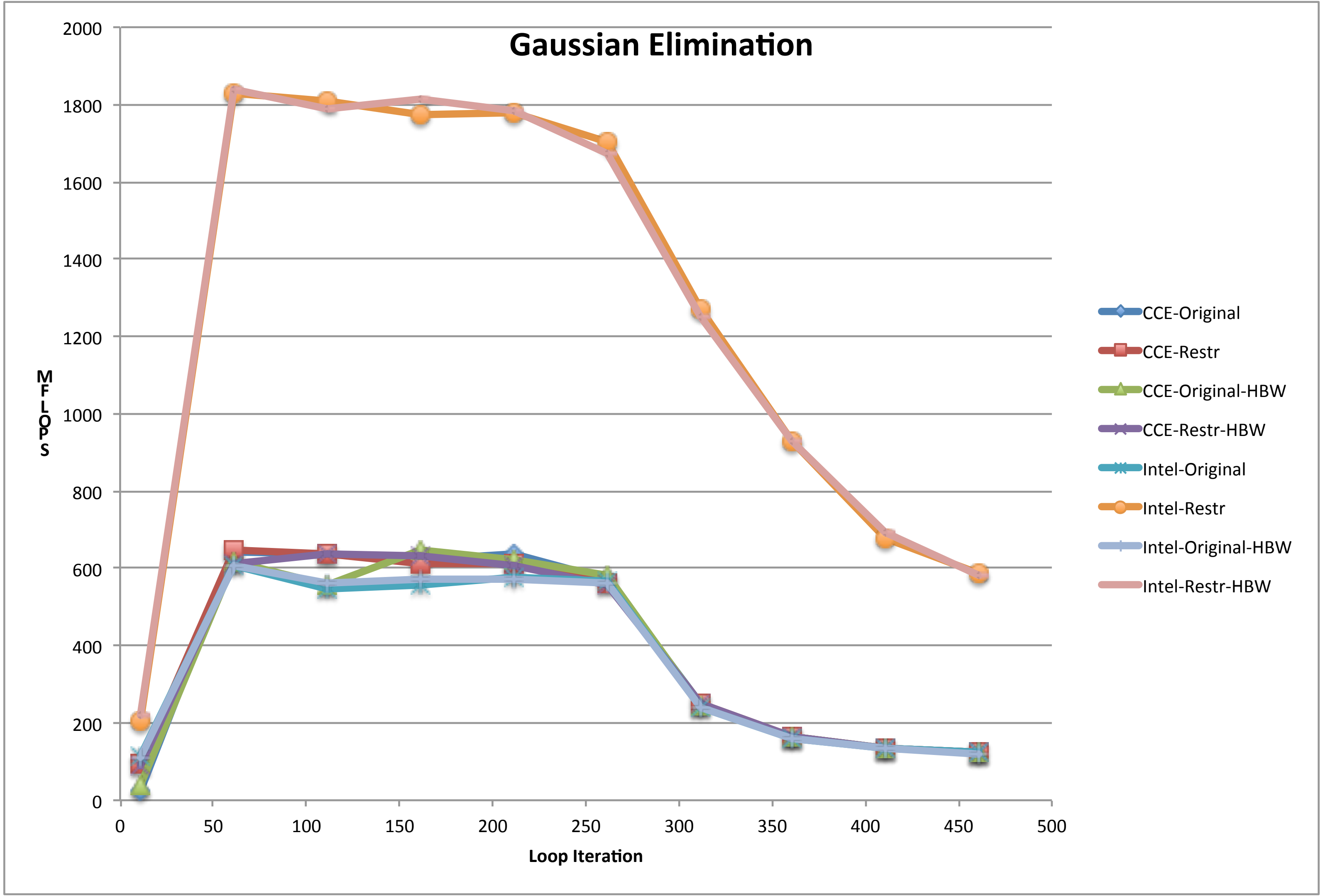
```
ORIGINAL
   43.    1             C  GAUSS ELIMINATION
   44.    1
   45.  + 1 2-----------<       DO 43020 I = 1, MATDIM
   46.    1 2                    A(I,I) = 1. / A(I,I)
   47.    1 2 r2--------<        DO 43020 J = I+1, MATDIM
   48.    1 2 r2                  A(J,I) = A(J,I) * A(I,I)
   49.    1 2 r2 Vr2----<         DO 43020 K = I+1, MATDIM
   50.    1 2 r2 Vr2               A(J,K) = A(J,K) - A(J,I) * A(I,K)
   51.    1 2 r2 Vr2-->>> 43020 CONTINUE

RESTRUCTURED
   83.  + 1 2-----------<       DO 43021 I = 1, MATDIM
   84.    1 2                    A(I,I) = 1. / A(I,I)
   85.    1 2 r2--------<        DO 43021 J = I+1, MATDIM
   86.    1 2 r2                  A(J,I) = A(J,I) * A(I,I)
   88.    1 2 r2          CDIR$ IVDEP
   90.    1 2 r2 Vr2----<         DO 43021 K = I+1, MATDIM
   91.    1 2 r2 Vr2               A(J,K) = A(J,K) - A(J,I) * A(I,K)
   92.    1 2 r2 Vr2-->>> 43021 CONTINUE
```

Gaussian Elimination

```
LOOP BEGIN at lp43020.f(45,10)
     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  line 46 and  line 50

     LOOP BEGIN at lp43020.f(47,11)
     <Distributed chunk1>
        remark #25426: Loop Distributed (2 way)
        remark #15335: loop was not vectorized: vectorization possible but seems inefficient.
        Use vector always directive or -vec-threshold0 to override
        remark #25439: unrolled with remainder by 8
     LOOP END

     LOOP BEGIN at lp43020.f(47,11)
     <Distributed chunk2>
        remark #15335: loop was not vectorized: vectorization possible but seems inefficient.
            Use vector always directive or -vec-threshold0 to override

        LOOP BEGIN at lp43020.f(49,12)
           remark #15335: loop was not vectorized: vectorization possible but seems inefficient.
                          Use vector always directive or -vec-threshold0 to override
           remark #25439: unrolled with remainder by 2
        LOOP END

        LOOP BEGIN at lp43020.f(49,12)
        <Remainder>
        LOOP END
     LOOP END

     LOOP BEGIN at lp43020.f(47,11)
     <Remainder, Distributed chunk1>
        remark #25436: completely unrolled by 7
     LOOP END
  LOOP END
```

Impact of size of stride

```
LOOP BEGIN at lp43020.f(83,10)
     remark #15542: loop was not vectorized: inner loop was already vectorized

     LOOP BEGIN at lp43020.f(85,11)
        remark #15301: OUTER LOOP WAS VECTORIZED

        LOOP BEGIN at lp43020.f(90,12)
           remark #15335: loop was not vectorized: vectorization possible but seems inefficient.
                                      Use vector always directive or -vec-threshold0 to override
        LOOP END
     LOOP END
  LOOP END
```


Gaussian Elimination
MFLOPS
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Recursion on One of Two dimensions

```
47.    1              C      THE ORIGINAL
48.    1
49.    1 V----------<       DO 43100 J = 1, N
50.    1 V                   AH = B(J) - B(J-1)
51.    1 V r4-------<        DO 43100 I = 2, N
52.    1 V r4                 A(I,J) = AH * A(I-1,J) + C(I,J)
53.    1 V r4------>> 43100 CONTINUE
```

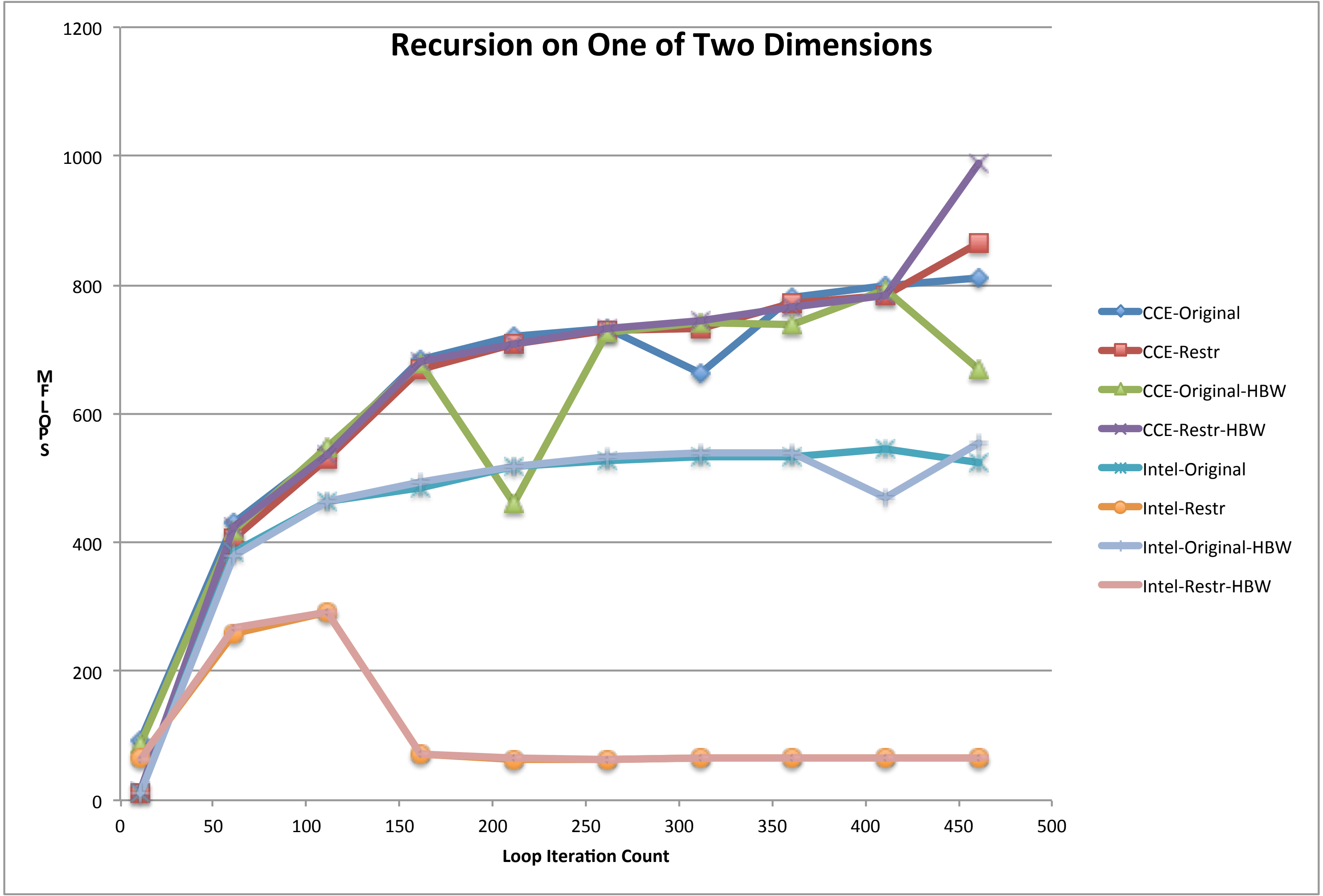
```
76.    1              C      THE RESTRUCTURED
77.    1
78.    1 Vr2--------<       DO 43101 J = 1, N
79.    1 Vr2                 VAH(J) = B(J) - B(J-1)
80.    1 Vr2--------> 43101 CONTINUE
81.    1
82.    1 ir4--------<       DO 43102 I = 2, N
83.    1 ir4 iV-----<        DO 43102 J = 1, N
84.    1 ir4 iV               A(I,J) = VAH(J) * A(I-1,J) + C(I,J)
85.    1 ir4 iV---->> 43102 CONTINUE
```

Recursion on One of Two dimensions

```
LOOP BEGIN at lp43100.f(49,10)
     remark #15301: OUTER LOOP WAS VECTORIZED

     LOOP BEGIN at lp43100.f(51,11)
        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 FLOW dependence between  line 52 and  line 52
     LOOP END
  LOOP END

  LOOP BEGIN at lp43100.f(49,10)
  <Remainder loop for vectorization>
     remark #15301: REMAINDER LOOP WAS VECTORIZED

     LOOP BEGIN at lp43100.f(51,11)
        remark #25460: No loop optimizations reported
     LOOP END
  LOOP END
```

Recursion on One of Two dimensions

```
LOOP BEGIN at lp43100.f(78,10)
<Peeled loop for vectorization>
   remark #15301: PEEL LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at lp43100.f(78,10)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at lp43100.f(78,10)
<Remainder loop for vectorization>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at lp43100.f(82,10)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at lp43100.f(83,11)
      remark #15300: LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at lp43100.f(83,11)
   <Remainder loop for vectorization>
      remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient.
                             Use vector always directive or -vec-threshold0 to override
   LOOP END
```


Recursion on One of Two Dimensions
MFLOPS
1200
1000
800
600
400
200
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Alternating Direction Implicit

```
48.       1              C      THE ORIGINAL
49.       1
50.       1 V----------<       DO 43111 J = 2, N
51.       1 V                   AH = B(J) - B(J-1)
52.       1 V r4-------<        DO 43110 I = 2, N
53.       1 V r4                 A(I,J) = AH  *  A(I-1,J) + C(I,J)
54.       1 V r4-------> 43110  CONTINUE
55.       1 V
56.       1 V                   BH  =  D(J) - D(J-1)
57.       1 V r4-------<        DO 43112 I  =  N, 2,  - 1
58.       1 V r4                 A(I,J)  =  BH  *  A(I+1,J) + C(I,J)
59.       1 V r4-------> 43112  CONTINUE
60.       1 V
61.       1 V----------> 43111 CONTINUE
```

Alternating Direction Implicit

```
 85.     1           C      THE  RESTRUCTURED
 86.     1
 87.     1 fV---------<      DO 43113 J = 2, N
 88.     1 fV                 VAH(J) = B(J) - B(J-1)
 89.     1 fV---------> 43113 CONTINUE
 90.     1
 91.     1 ir4--------<      DO 43114 I = 2, N
 92.     1 ir4 fi-----<       DO 43114 J  =  2, N
 93.     1 ir4 fi              A(I,J) = VAH(J)  *  A(I-1,J) + C(I,J)
 94.     1 ir4 fi---->> 43114 CONTINUE
 95.     1
 96.     1 f----------<      DO 43115 J  =  2, N
 97.     1 f                  VBH(J)  =  D(J) - D(J-1)
 98.     1 f----------> 43115 CONTINUE
 99.     1
100.     1 ir4--------<      DO 43116 I  =  N, 2,  - 1
101.     1 ir4 iV-----<       DO 43116 J  =  2, N
102.     1 ir4 iV              A(I,J)  =  VBH(J)  *  A(I+1,J) + C(I,J)
103.     1 ir4 iV---->> 43116 CONTINUE
```

Alternating Direction Implicit

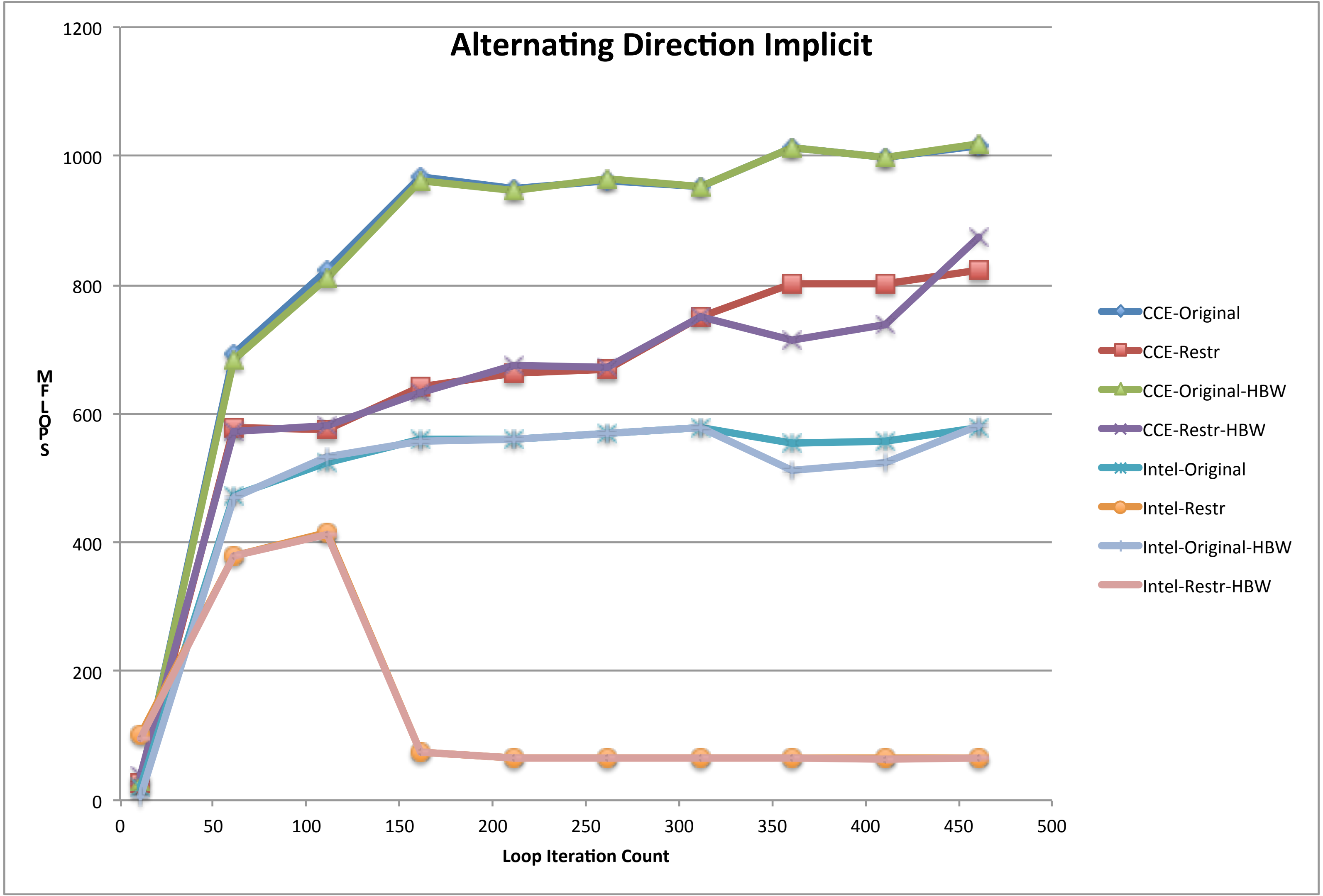
```
LOOP BEGIN at lp43110.f(50,10)
     remark #15301: OUTER LOOP WAS VECTORIZED

     LOOP BEGIN at lp43110.f(52,11)
        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 FLOW dependence between  line 53 and  line 53
     LOOP END

     LOOP BEGIN at lp43110.f(57,11)
        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 FLOW dependence between  line 58 and  line 58
     LOOP END
  LOOP END

  LOOP BEGIN at lp43110.f(50,10)
  <Remainder loop for vectorization>
     remark #15301: REMAINDER LOOP WAS VECTORIZED

     LOOP BEGIN at lp43110.f(52,11)
        remark #25460: No loop optimizations reported
     LOOP END

     LOOP BEGIN at lp43110.f(57,11)
        remark #25460: No loop optimizations reported
     LOOP END
  LOOP END
```

Alternating Direction Implicit

```
LOOP BEGIN at lp43110.f(87,10)
  <Peeled loop for vectorization>
     remark #15301: PEEL LOOP WAS VECTORIZED
  LOOP END

  LOOP BEGIN at lp43110.f(87,10)
     remark #15300: LOOP WAS VECTORIZED
  LOOP END

  LOOP BEGIN at lp43110.f(87,10)
  <Remainder loop for vectorization>
     remark #15301: REMAINDER LOOP WAS VECTORIZED
  LOOP END
   LOOP BEGIN at lp43110.f(91,10)
     remark #15542: loop was not vectorized: inner loop was already vectorized

     LOOP BEGIN at lp43110.f(92,11)
        remark #15300: LOOP WAS VECTORIZED
     LOOP END

     LOOP BEGIN at lp43110.f(92,11)
     <Remainder loop for vectorization>
        remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient.
                       Use vector always directive or -vec-threshold0 to override
     LOOP END
  LOOP END
```

Alternating Direction Implicit

```
LOOP BEGIN at lp43110.f(96,10)
<Peeled loop for vectorization>
   remark #15301: PEEL LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at lp43110.f(96,10)
   remark #15300: LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at lp43110.f(96,10)
<Remainder loop for vectorization>
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at lp43110.f(100,10)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at lp43110.f(101,11)
      remark #15300: LOOP WAS VECTORIZED
   LOOP END

   LOOP BEGIN at lp43110.f(101,11)
   <Remainder loop for vectorization>
      remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient. Use v
   LOOP END
LOOP END
```


Alternating Direction Implicit
M F L O P S
0
200
400
600
800
1000
1200
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Strongly Implicit Procedure

```
51.    1                C      THE ORIGINAL
  52.    1
  53. + 1 2------------<        DO 43140 J = 2, N
  54. + 1 2 r2---------<         DO 43140 I = 2, N
  55.    1 2 r2                   A(I,J,1) = A(I,J,1) - B(I,J) * A(I-1,J,1)
  56.    1 2 r2                *                      - C(I,J) * A(I,J-1,1)
  57.    1 2 r2                   A(I,J,2) = A(I,J,2) - B(I,J) * A(I-1,J,2)
  58.    1 2 r2                *                      - C(I,J) * A(I,J-1,2)
  59.    1 2 r2                   A(I,J,3) = A(I,J,3) - B(I,J) * A(I-1,J,3)
  60.    1 2 r2                *                      - C(I,J) * A(I,J-1,3)
  61.    1 2 r2-------->> 43140 CONTINUE
```

Strongly Implicit Procedure

```
 89.    1                C      THE RESTRUCTURED
 90.    1
 91.    1                      NDIAGS = 2 * N - 3
 92.    1                      ISTART = 1
 93.    1                      JSTART = 2
 94.    1                      LDIAG  = 0
 95.  + 1 2------------<       DO 43141 IDIAGS = 1, NDIAGS
 96.    1 2                     IF(IDIAGS .LE. N-1 ) THEN
 97.    1 2                        ISTART = ISTART + 1
 98.    1 2                        LDIAG  = LDIAG  + 1
 99.    1 2                     ELSE
100.    1 2                        JSTART = JSTART + 1
101.    1 2                        LDIAG  = LDIAG  - 1
102.    1 2                     ENDIF
103.    1 2                     I = ISTART + 1
104.    1 2                     J = JSTART - 1
105.    1 2              CDIR$ IVDEP
106.    1 2              CVD$ NODEPCHK
107.    1 2              *VDIR NODEP
108.    1 2 Vr2--------<        DO 43142 IPOINT = 1, LDIAG
109.    1 2 Vr2                  I = I - 1
110.    1 2 Vr2                  J = J + 1
111.    1 2 Vr2                  A(I,J,1) = A(I,J,1) - B(I,J) * A(I-1,J,1)
112.    1 2 Vr2               *                      - C(I,J) * A(I,J-1,1)
113.    1 2 Vr2                  A(I,J,2) = A(I,J,2) - B(I,J) * A(I-1,J,2)
114.    1 2 Vr2               *                      - C(I,J) * A(I,J-1,2)
115.    1 2 Vr2                  A(I,J,3) = A(I,J,3) - B(I,J) * A(I-1,J,3)
116.    1 2 Vr2               *                      - C(I,J) * A(I,J-1,3)
117.    1 2 Vr2--------> 43142  CONTINUE
118.    1 2------------> 43141 CONTINUE
```

Strongly Implicit Procedure

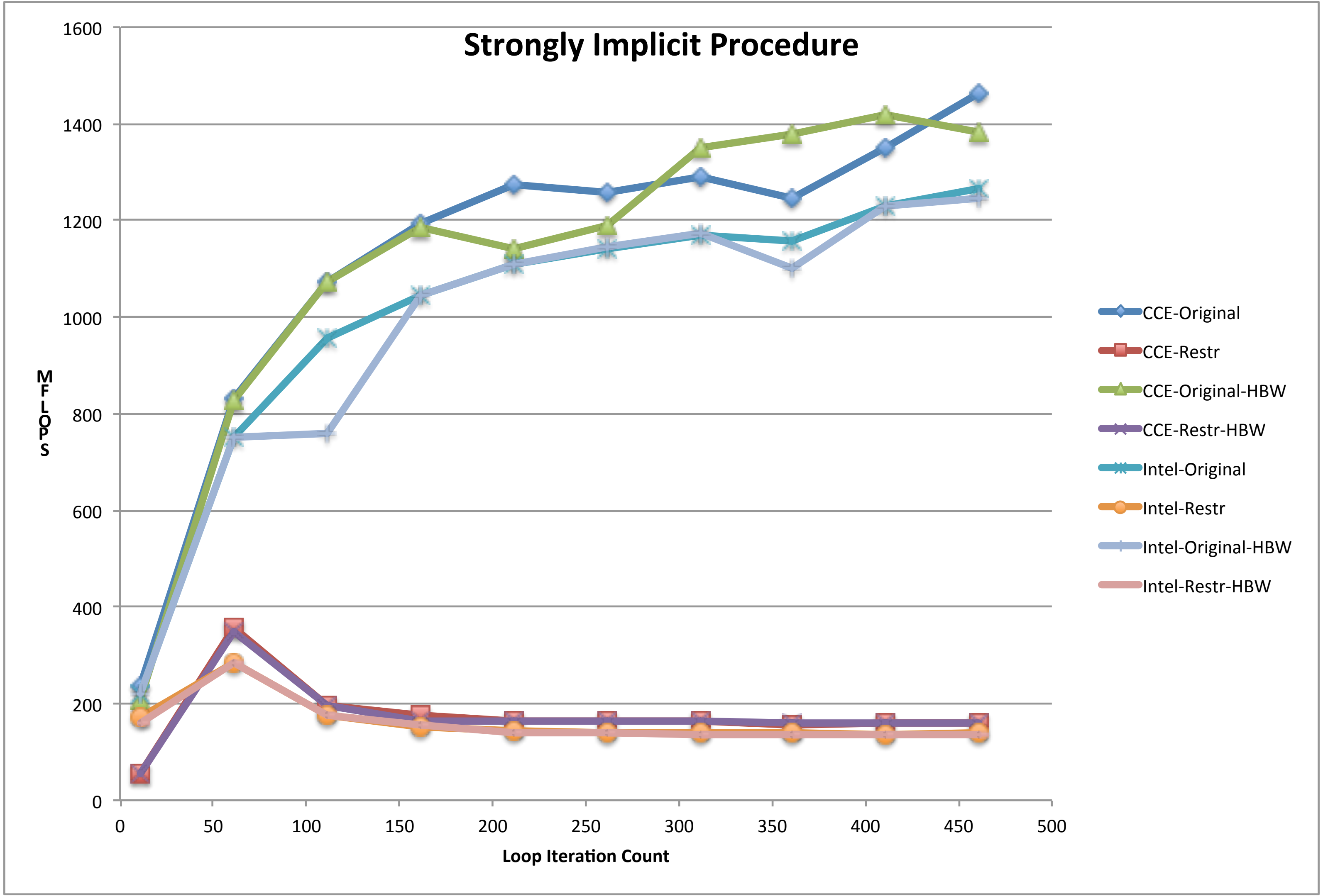
```
 89.    1                C      THE RESTRUCTURED
 90.    1
 91.    1                      NDIAGS = 2 * N - 3
 92.    1                      ISTART = 1
 93.    1                      JSTART = 2
 94.    1                      LDIAG   = 0
 95.  + 1 2------------<       DO 43141 IDIAGS = 1, NDIAGS
 96.    1 2                     IF(IDIAGS .LE. N-1 ) THEN
 97.    1 2                        ISTART = ISTART + 1
 98.    1 2                        LDIAG  = LDIAG   + 1
 99.    1 2                     ELSE
100.    1 2                        JSTART = JSTART + 1
101.    1 2                        LDIAG  = LDIAG   - 1
102.    1 2                     ENDIF
103.    1 2                     I = ISTART + 1
104.    1 2                     J = JSTART - 1
105.    1 2              CDIR$ IVDEP
106.    1 2              CVD$ NODEPCHK
107.    1 2              *VDIR NODEP
108.    1 2 Vr2--------<        DO 43142 IPOINT = 1, LDIAG
109.    1 2 Vr2                  I = I - 1
110.    1 2 Vr2                  J = J + 1
111.    1 2 Vr2                  A(I,J,1) = A(I,J,1) - B(I,J) * A(I-1,J,1)
112.    1 2 Vr2               *                      - C(I,J) * A(I,J-1,1)
113.    1 2 Vr2                  A(I,J,2) = A(I,J,2) - B(I,J) * A(I-1,J,2)
114.    1 2 Vr2               *                      - C(I,J) * A(I,J-1,2)
115.    1 2 Vr2                  A(I,J,3) = A(I,J,3) - B(I,J) * A(I-1,J,3)
116.    1 2 Vr2               *                      - C(I,J) * A(I,J-1,3)
117.    1 2 Vr2--------> 43142  CONTINUE
118.    1 2------------> 43141 CONTINUE
```

Strongly Implicit Procedure

```
LOOP BEGIN at lp43140.f(53,10)
     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 FLOW dependence between  line 55 and  line 55

     LOOP BEGIN at lp43140.f(54,11)
        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 FLOW dependence between  line 55 and  line 55
        remark #25456: Number of Array Refs Scalar Replaced In Loop: 7
     LOOP END
  LOOP END
```

Strongly Implicit Procedure

```
LOOP BEGIN at lp43140.f(95,10)
     remark #15542: loop was not vectorized: inner loop was already vectorized

     LOOP BEGIN at lp43140.f(108,11)
        remark #15300: LOOP WAS VECTORIZED
     LOOP END

     LOOP BEGIN at lp43140.f(108,11)
     <Remainder loop for vectorization>
        remark #15335: remainder loop was not vectorized: vectorization possible but seems inefficient.
                         Use vector always directive or -vec-threshold0 to override
        remark #25456: Number of Array Refs Scalar Replaced In Loop: 4
     LOOP END
  LOOP END
```


Strongly Implicit Procedure
MFLOPS
1600
1400
1200
1000
800
600
400
200
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Loop Carried Dependent Scalar

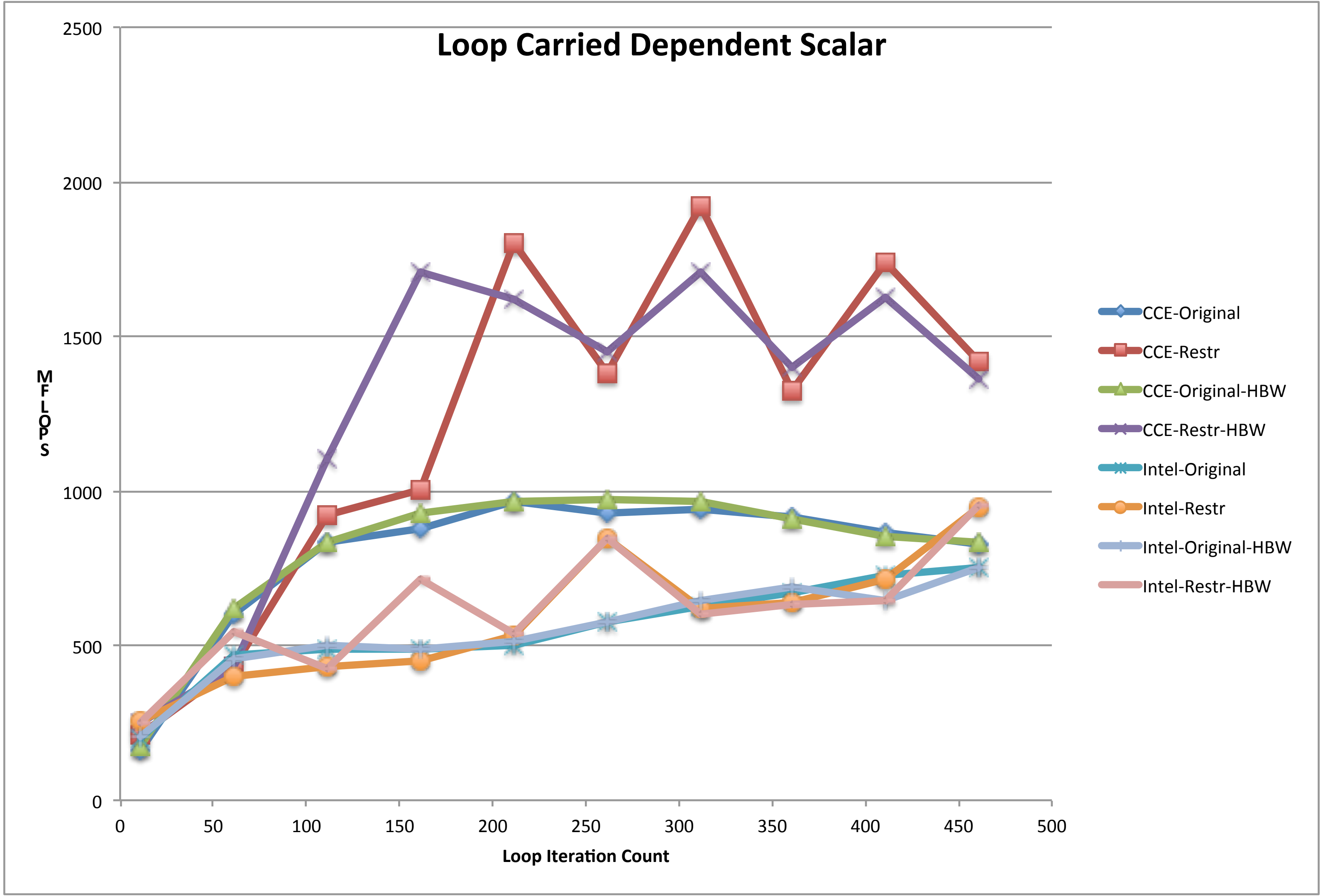
```
62.    1          C      THE ORIGINAL
63.    1
64.    1                PF = 0.0
65.  + 1 2------<       DO 44030 I = 2, N
66.    1 2               AV    = B(I) * RV
67.    1 2               PB    = PF
68.    1 2               PF    = C(I)
69.    1 2               IF ((D(I) + D(I+1)) .LT. 0.) PF = -C(I+1)
70.    1 2               AA    = E(I) - E(I-1) + F(I) - F(I-1)
71.    1 2              1      + G(I) + G(I-1) - H(I) - H(I-1)
72.    1 2               BB    = R(I) + S(I-1) + T(I) + T(I-1)
73.    1 2              1      - U(I) - U(I-1) + V(I) + V(I-1)
74.    1 2              2      - W(I) + W(I-1) - X(I) + X(I-1)
75.    1 2               A(I) = AV * (AA + BB + PF - PB + Y(I) - Z(I)) + A(I)
76.    1 2------> 44030 CONTINUE
77.    1
```

Loop Carried Dependent Scalar

```
 98.      1          C       THE RESTRUCTURED
 99.      1
100.      1                 VPF(1) = 0.0
101.      1 Vr2---<         DO 44031 I = 2, N
102.      1 Vr2              AV     = B(I) * RV
103.      1 Vr2              VPF(I) = C(I)
104.      1 Vr2              IF ((D(I) + D(I+1)) .LT. 0.) VPF(I) = -C(I+1)
105.      1 Vr2              AA    = E(I) - E(I-1) + F(I) - F(I-1)
106.      1 Vr2             1      + G(I) + G(I-1) - H(I) - H(I-1)
107.      1 Vr2              BB    = R(I) + S(I-1) + T(I) + T(I-1)
108.      1 Vr2             1      - U(I) - U(I-1) + V(I) + V(I-1)
109.      1 Vr2             2      - W(I) + W(I-1) - X(I) + X(I-1)
110.      1 Vr2              A(I)  = AV * (AA + BB + VPF(I) - VPF(I-1) + Y(I) - Z(I)) + A(I)
111.      1 Vr2---> 44031 CONTINUE   77.     1
```

Loop Carried Dependent Scalar

```
LOOP BEGIN at lp44030.f(65,10)
     remark #15344: loop was not vectorized: vector dependence prevents vectorization.
                Use level 5 report for details First dependence is shown below.
     remark #15346: vector dependence: assumed FLOW dependence between  line 68 and  line 67
     remark #25456: Number of Array Refs Scalar Replaced In Loop: 10
  LOOP END


LOOP BEGIN at lp44030.f(101,10)
     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 FLOW dependence between  line 104 and  line 110
     remark #25456: Number of Array Refs Scalar Replaced In Loop: 12
  LOOP END
```



Loop Carried Dependent Scalar
MFLOPS
0
500
1000
1500
2000
2500
Loop Iteration Count
0
50
100
150
200
250
300
350
400
450
500
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Complex Reduction

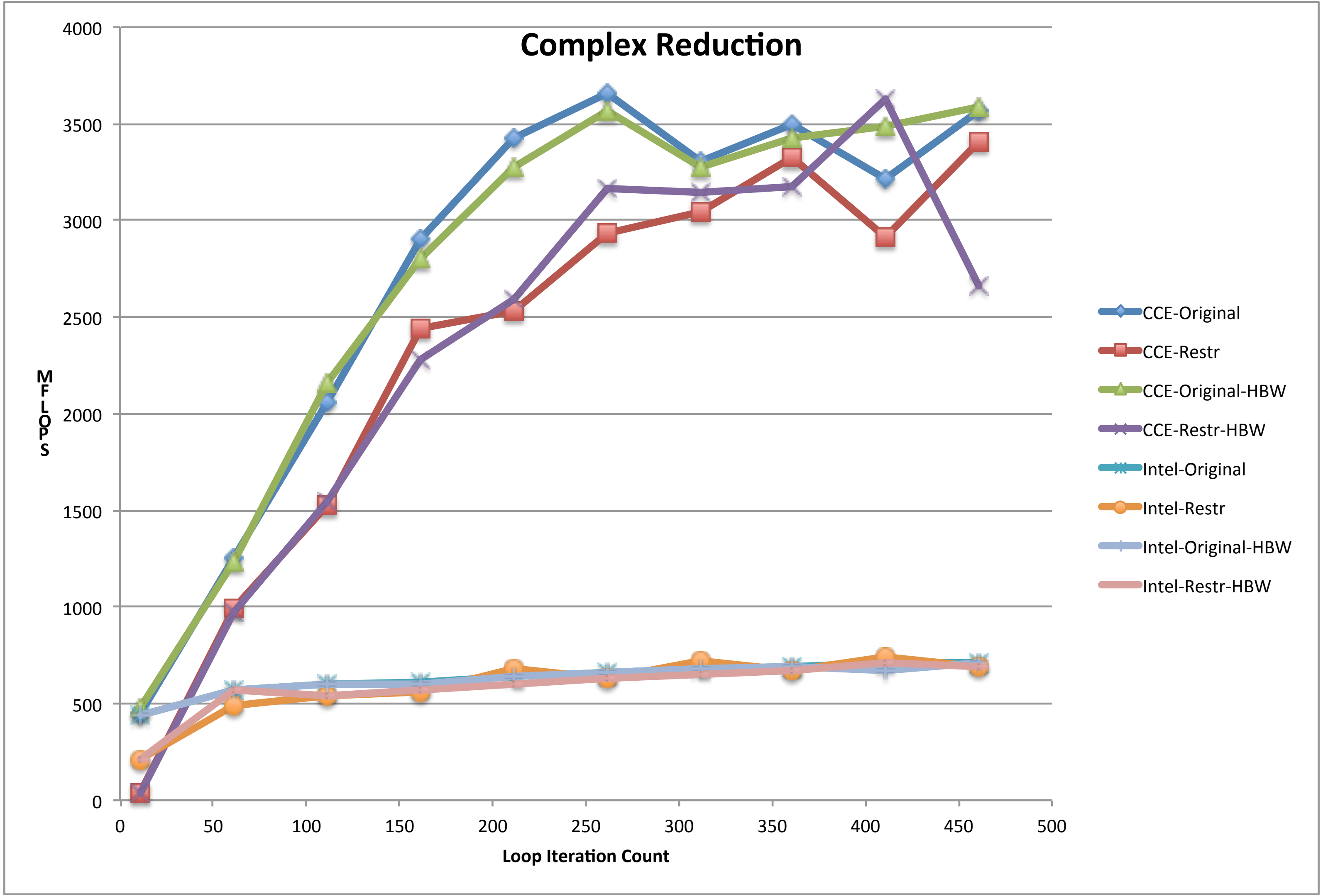
```
61.     1       C       THE ORIGINAL
62.     1
63.     1 V---<        DO 44040 I = 2, N
64.     1 V             RR          = 1. / A(I,1)
65.     1 V             U           = A(I,2) * RR
66.     1 V             V           = A(I,3) * RR
67.     1 V             W           = A(I,4) * RR
68.     1 V             SNDSP       = SQRT (GD * (A(I,5) * RR + .5* (U*U + V*V + W*W)))
69.     1 V             SIGA        = ABS (XT + U*B(I) + V*C(I) + W*D(I))
70.     1 V           *               + SNDSP * SQRT (B(I)**2 + C(I)**2 + D(I)**2)
71.     1 V             SIGB        = ABS (YT + U*E(I) + V*F(I) + W*G(I))
72.     1 V           *               + SNDSP * SQRT (E(I)**2 + F(I)**2 + G(I)**2)
73.     1 V             SIGC        = ABS (ZT + U*H(I) + V*R(I) + W*S(I))
74.     1 V           *               + SNDSP * SQRT (H(I)**2 + R(I)**2 + S(I)**2)
75.     1 V             SIGABC      = AMAX1 (SIGA, SIGB, SIGC)
76.     1 V             IF (SIGABC.GT.SIGMAX) THEN
77.     1 V             IMAX        = I
78.     1 V             SIGMAX      = SIGABC
79.     1 V             ENDIF
80.     1 V---> 44040 CONTINUE
```

Complex Reduction

```
99.    1       C      THE RESTRUCTURED
100.    1
101.    1 fV--<       DO 44041 I = 2, N
102.    1 fV           RR         = 1. / A(I,1)
103.    1 fV           U          = A(I,2) * RR
104.    1 fV           V          = A(I,3) * RR
105.    1 fV           W          = A(I,4) * RR
106.    1 fV           SNDSP      = SQRT (GD * (A(I,5) * RR + .5* (U*U + V*V + W*W)))
107.    1 fV           SIGA       = ABS (XT + U*B(I) + V*C(I) + W*D(I))
108.    1 fV         *              + SNDSP * SQRT (B(I)**2 + C(I)**2 + D(I)**2)
109.    1 fV           SIGB       = ABS (YT + U*E(I) + V*F(I) + W*G(I))
110.    1 fV         *              + SNDSP * SQRT (E(I)**2 + F(I)**2 + G(I)**2)
111.    1 fV           SIGC       = ABS (ZT + U*H(I) + V*R(I) + W*S(I))
112.    1 fV         *              + SNDSP * SQRT (H(I)**2 + R(I)**2 + S(I)**2)
113.    1 fV           VSIGABC(I) = AMAX1 (SIGA, SIGB, SIGC)
114.    1 fV--> 44041 CONTINUE
115.    1
116.    1 f---<       DO 44042 I = 2, N
117.    1 f            IF (VSIGABC(I) .GT. SIGMAX) THEN
118.    1 f             IMAX       = I
119.    1 f             SIGMAX     = VSIGABC(I)
120.    1 f            ENDIF
121.    1 f---> 44042 CONTINUE
```

Complex Reduction

```
LOOP BEGIN at lp44040.f(63,10)
     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 ANTI dependence between  line 76 and  line 78
     remark #25456: Number of Array Refs Scalar Replaced In Loop: 9
  LOOP END

LOOP BEGIN at lp44040.f(101,10)
     remark #25045: Fused Loops: ( 101 116 )

     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 ANTI dependence between  line 117 and  line 119
     remark #25456: Number of Array Refs Scalar Replaced In Loop: 9
  LOOP END
```


Complex Reduction
MFLOPS
0
500
1000
1500
2000
2500
3000
3500
4000
Loop Iteration Count
0
50
100
150
200
250
300
350
400
450
500
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Matrix Multiply

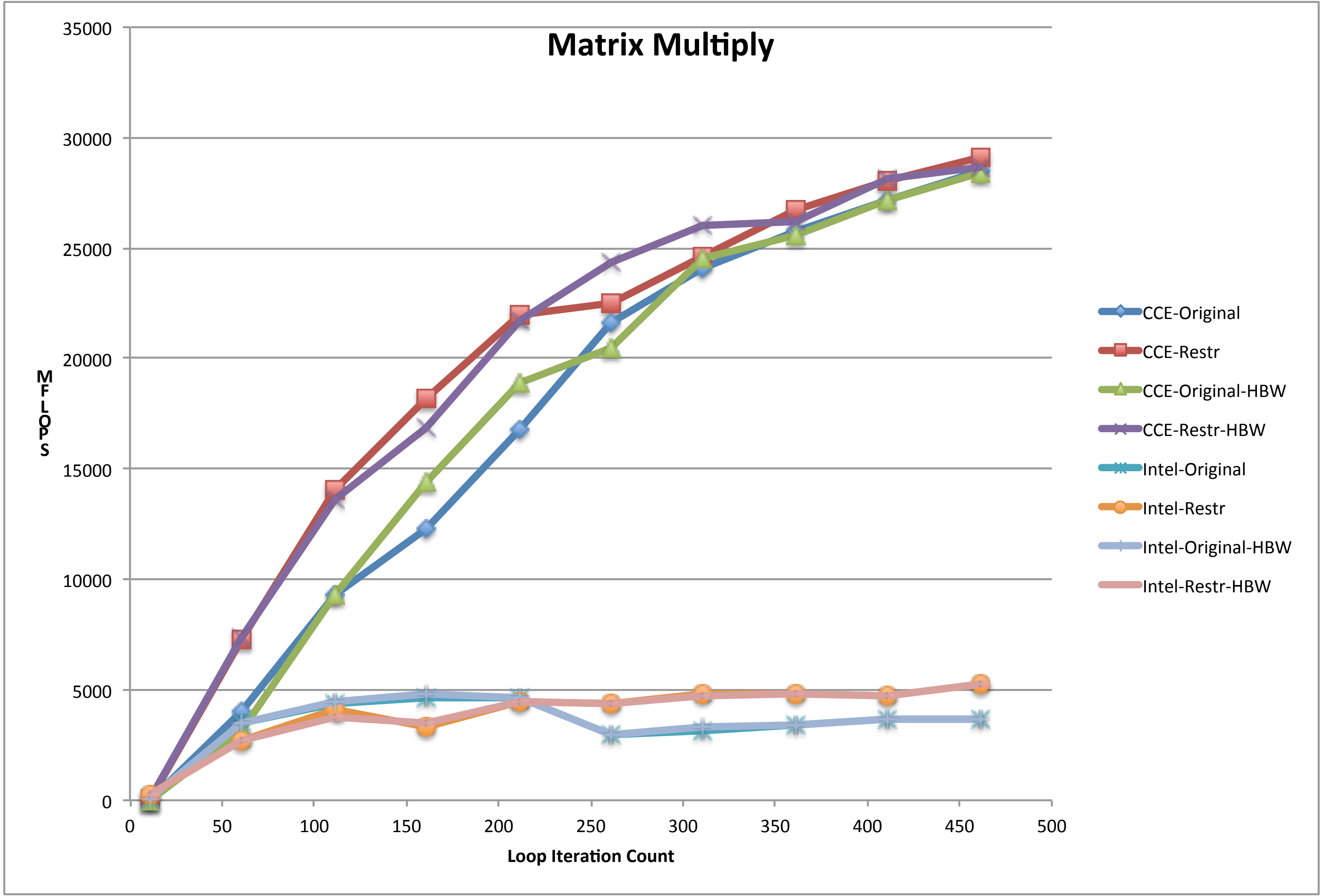
```
44.    1
45.    1              C      THE ORIGINAL
46.    1
47.  + 1 2----------<       DO 44050 I = 1, N
48.  + 1 2 3--------<        DO 44050 J = 1, N
49.    1 2 3                  A(I,J) = 0.0
50.  + 1 2 3 4------<         DO 44050 K = 1, N
51.    1 2 3 4 A---<>          A(I,J) = A(I,J) + B(I,K) * C(K,J)
52.    1 2 3 4---->>> 44050 CONTINUE

77.    1
78.    1              C      THE RESTRUCTURED
79.    1
80.  + 1 2----------<       DO 44051 J = 1, N
81.    1 2 A--------<        DO 44051 I = 1, N
82.    1 2 A                  A(I,J) = 0.0
83.    1 2 A------->> 44051 CONTINUE
84.    1
85.  + 1 2----------<       DO 44052 K = 1, N
86.  + 1 2 3--------<        DO 44052 J = 1, N
87.  + 1 2 3 4------<         DO 44052 I = 1, N
88.    1 2 3 4 A---<>          A(I,J) = A(I,J) + B(I,K) * C(K,J)
89.    1 2 3 4---->>> 44052 CONTINUE
90.    1              C
```

Matrix Multiply

```
LOOP BEGIN at lp44050.f(47,10)
           <Remainder, Distributed chunk2>
              remark #15542: loop was not vectorized: inner loop was already vectorized

              LOOP BEGIN at lp44050.f(48,11)
              <Distributed chunk2>
                 remark #15542: loop was not vectorized: inner loop was already vectorized

                 LOOP BEGIN at lp44050.f(50,12)
                 <Peeled loop for vectorization>
                    remark #15335: peel loop was not vectorized: vectorization possible but seems inefficient.
                                 Use vector always directive or -vec-threshold0 to override
                 LOOP END

                 LOOP BEGIN at lp44050.f(50,12)
                    remark #25085: Preprocess Loopnests: Moving Out Load and Store    [ lp44050.f(51,10) ]
                    remark #15300: LOOP WAS VECTORIZED
                 LOOP END

                 LOOP BEGIN at lp44050.f(50,12)
                 <Remainder loop for vectorization>
                    remark #15301: REMAINDER LOOP WAS VECTORIZED
                 LOOP END
              LOOP END
           LOOP END
```

Matrix Multiply

```
LOOP BEGIN at lp44050.f(80,10)
     remark #15542: loop was not vectorized: inner loop was already vectorized

     LOOP BEGIN at lp44050.f(81,11)
     <Peeled loop for vectorization>
        remark #15335: peel loop was not vectorized: vectorization possible but seems inefficient.
                    Use vector always directive or -vec-threshold0 to override
     LOOP END

     LOOP BEGIN at lp44050.f(81,11)
        remark #15300: LOOP WAS VECTORIZED
     LOOP END

     LOOP BEGIN at lp44050.f(81,11)
     <Remainder loop for vectorization>
        remark #15301: REMAINDER LOOP WAS VECTORIZED
     LOOP END
  LOOP END
```


Matrix Multiply
MFLOPS
35000
30000
25000
20000
15000
10000
5000
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Vector Versus Reduction

```
 62.    1 2                  C      THE ORIGINAL
 63.    1 2
 64.    1 2                        J = 1
 65.  + 1 2 3--------------<       DO 45011 I = 1, IM
 66.    1 2 3                       SUM1    = 0.0
 67.    1 2 3                       SUM2    = 0.0
 68.    1 2 3                       SUM4    = 0.0
 69.    1 2 3                       SUM5    = 0.0
 70.    1 2 3 Vr2----------<        DO 45010 K = 2, KM
 71.    1 2 3 Vr2                    KK   = KM - K + 1
 72.    1 2 3 Vr2
 73.    1 2 3 Vr2                    SUM1 = SUM1 + 4.0 * ( A(J+1,K ,I,1) * A(J+1,K ,I,6)
 74.    1 2 3 Vr2                 *              +         A(J+1,KK,I,1) * A(J+1,KK,I,6) )
 75.    1 2 3 Vr2                 *              -         A(J+2,K ,I,1) * A(J+2,K ,I,6)
 76.    1 2 3 Vr2                 *              -         A(J+2,KK,I,1) * A(J+2,KK,I,6)
 77.    1 2 3 Vr2
 78.    1 2 3 Vr2                    SUM2 = SUM2 + 4.0 * ( A(J+1,K, I,2) * A(J+1,K, I,6)
 79.    1 2 3 Vr2                 *              +         A(J+1,KK,I,2) * A(J+1,KK,I,6) )
 80.    1 2 3 Vr2                 *              -         A(J+2,K, I,2) * A(J+2,K, I,6)
 81.    1 2 3 Vr2                 *              -         A(J+2,KK,I,2) * A(J+2,KK,I,6)
 82.    1 2 3 Vr2
 83.    1 2 3 Vr2                    SUM4 = SUM4 + 4.0 * ( A(J+1,K, I,4) * A(J+1,K, I,6)
 84.    1 2 3 Vr2                 *              +         A(J+1,KK,I,4) * A(J+1,KK,I,6) )
 85.    1 2 3 Vr2                 *              -         A(J+2,K, I,4) * A(J+2,K, I,6)
 86.    1 2 3 Vr2                 *              -         A(J+2,KK,I,4) * A(J+2,KK,I,6)
 87.    1 2 3 Vr2
 88.    1 2 3 Vr2                    SUM5 = SUM5 + 4.0 * ( A(J+1,K, I,5) * A(J+1,K, I,6)
 89.    1 2 3 Vr2                 *              +         A(J+1,KK,I,5) * A(J+1,KK,I,6) )
 90.    1 2 3 Vr2                 *              -         A(J+2,K, I,5) * A(J+2,K, I,6)
 91.    1 2 3 Vr2                 *              -         A(J+2,KK,I,5) * A(J+2,KK,I,6)
 92.    1 2 3 Vr2----------> 45010    CONTINUE
 93.    1 2 3
 94.    1 2 3 Vr2----------<        DO 45011 K = 2, KM
 95.    1 2 3 Vr2                    A(J,K,I,1) = SUM1 / (6.0 * (KM-2) * A(J,K,I,6))
 96.    1 2 3 Vr2                    A(J,K,I,2) = SUM2 / (6.0 * (KM-2) * A(J,K,I,6))
 97.    1 2 3 Vr2                    A(J,K,I,3) = 0.0
 98.    1 2 3 Vr2                    A(J,K,I,4) = SUM4 / (6.0 * (KM-2) * A(J,K,I,6))
 99.    1 2 3 Vr2                    A(J,K,I,5) = SUM5 / (6.0 * (KM-2) * A(J,K,I,6))
100.    1 2 3 Vr2--------->> 45011 CONTINUE
```

Vector Versus Reduction

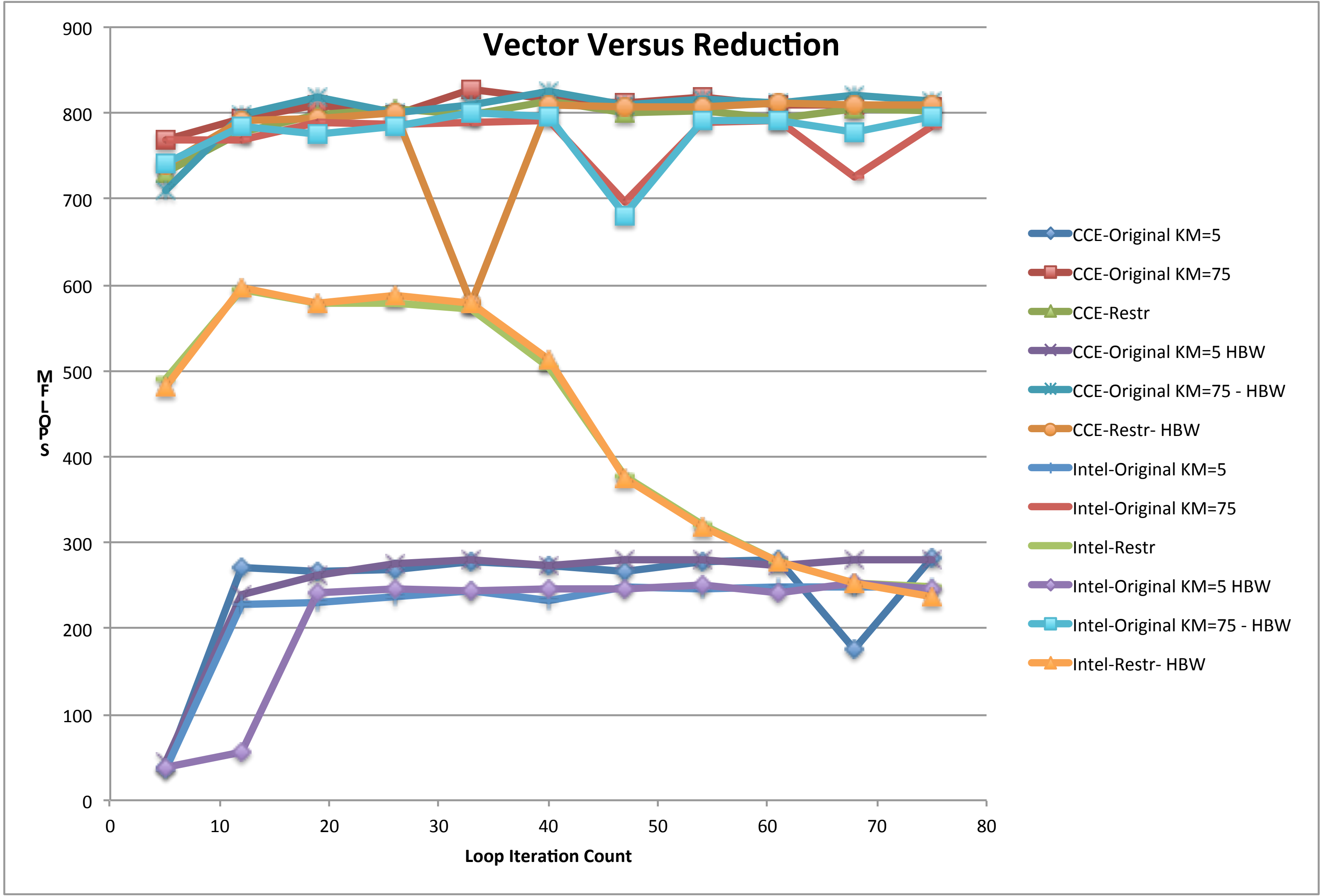
```
139.    1 2 iVr2------------<      DO 45013  K = 2, KM
140.    1 2 iVr2                    KK         = KM - K + 1
141.  + 1 2 iVr2 fi---------<       DO 45013 I = 1, IM
142.    1 2 iVr2 fi                  VSUM1(I) = VSUM1(I) + 4.0 * ( A(J+1,K ,I,1) * A(J+1,K ,I,6)
143.    1 2 iVr2 fi               *                      +         A(J+1,KK,I,1) * A(J+1,KK,I,6) )
144.    1 2 iVr2 fi               *                      -         A(J+2,K ,I,1) * A(J+2,K ,I,6)
145.    1 2 iVr2 fi               *                      -         A(J+2,KK,I,1) * A(J+2,KK,I,6)
146.    1 2 iVr2 fi
147.    1 2 iVr2 fi                  VSUM2(I) = VSUM2(I) + 4.0 * ( A(J+1,K, I,2) * A(J+1,K, I,6)
148.    1 2 iVr2 fi               *                      +         A(J+1,KK,I,2) * A(J+1,KK,I,6) )
149.    1 2 iVr2 fi               *                      -         A(J+2,K, I,2) * A(J+2,K, I,6)
150.    1 2 iVr2 fi               *                      -         A(J+2,KK,I,2) * A(J+2,KK,I,6)
151.    1 2 iVr2 fi
152.    1 2 iVr2 fi                  VSUM4(I) = VSUM4(I) + 4.0 * ( A(J+1,K, I,4) * A(J+1,K, I,6)
153.    1 2 iVr2 fi               *                      +         A(J+1,KK,I,4) * A(J+1,KK,I,6) )
154.    1 2 iVr2 fi               *                      -         A(J+2,K, I,4) * A(J+2,K, I,6)
155.    1 2 iVr2 fi               *                      -         A(J+2,KK,I,4) * A(J+2,KK,I,6)
156.    1 2 iVr2 fi
157.    1 2 iVr2 fi                  VSUM5(I) = VSUM5(I) + 4.0 * ( A(J+1,K, I,5) * A(J+1,K, I,6)
158.    1 2 iVr2 fi               *                      +         A(J+1,KK,I,5) * A(J+1,KK,I,6) )
159.    1 2 iVr2 fi               *                      -         A(J+2,K, I,5) * A(J+2,K, I,6)
160.    1 2 iVr2 fi               *                      -         A(J+2,KK,I,5) * A(J+2,KK,I,6)
161.    1 2 iVr2 fi
162.    1 2 iVr2 fi------->> 45013 CONTINUE
163.    1 2
164.    1 2 iVr2------------<      DO 45014  K = 2, KM
165.  + 1 2 iVr2 fi---------<       DO 45014 I = 1, IM
166.    1 2 iVr2 fi                  A(J,K,I,1) = VSUM1(I) / (6.0 * (KM-2) * A(J,K,I,6))
167.    1 2 iVr2 fi                  A(J,K,I,2) = VSUM2(I) / (6.0 * (KM-2) * A(J,K,I,6))
168.    1 2 iVr2 fi                  A(J,K,I,3) = 0.0
169.    1 2 iVr2 fi                  A(J,K,I,4) = VSUM4(I) / (6.0 * (KM-2) * A(J,K,I,6))
170.    1 2 iVr2 fi                  A(J,K,I,5) = VSUM5(I) / (6.0 * (KM-2) * A(J,K,I,6))
171.    1 2 iVr2 fi------->> 45014 CONTINUE
```

Vector Versus Reduction

```
LOOP BEGIN at lp45010.f(65,10)
        remark #15542: loop was not vectorized: inner loop was already vectorized

        LOOP BEGIN at lp45010.f(70,11)
           remark #15300: LOOP WAS VECTORIZED
        LOOP END

        LOOP BEGIN at lp45010.f(70,11)
        <Remainder loop for vectorization>
           remark #15301: REMAINDER LOOP WAS VECTORIZED
        LOOP END

        LOOP BEGIN at lp45010.f(94,11)
           remark #15300: LOOP WAS VECTORIZED
        LOOP END

        LOOP BEGIN at lp45010.f(94,11)
        <Remainder loop for vectorization>
           remark #15301: REMAINDER LOOP WAS VECTORIZED
        LOOP END
     LOOP END
```

Vector Versus Reduction

```
LOOP BEGIN at lp45010.f(139,10)
        remark #15542: loop was not vectorized: inner loop was already vectorized

        LOOP BEGIN at lp45010.f(141,11)
           remark #15300: LOOP WAS VECTORIZED
        LOOP END

        LOOP BEGIN at lp45010.f(141,11)
        <Remainder loop for vectorization>
           remark #15301: REMAINDER LOOP WAS VECTORIZED
        LOOP END
     LOOP END

     LOOP BEGIN at lp45010.f(164,10)
        remark #15542: loop was not vectorized: inner loop was already vectorized

        LOOP BEGIN at lp45010.f(165,11)
           remark #15300: LOOP WAS VECTORIZED
        LOOP END

        LOOP BEGIN at lp45010.f(165,11)
        <Remainder loop for vectorization>
           remark #15301: REMAINDER LOOP WAS VECTORIZED
        LOOP END
     LOOP END
```


Vector Versus Reduction
MFLOPS
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
Loop Iteration Count
CCE-Original KM=5
CCE-Original KM=75
CCE-Restr
CCE-Original KM=5 HBW
CCE-Original KM=75 - HBW
CCE-Restr- HBW
Intel-Original KM=5
Intel-Original KM=75
Intel-Restr
Intel-Original KM=5 HBW
Intel-Original KM=75 - HBW
Intel-Restr- HBW

Rectangular Matrix Multiply

```
44.    1
45.    1             C      THE ORIGINAL
46.    1
47.  + 1 2---------<       DO 46020 I = 1,N
48.  + 1 2 3-------<        DO 46020 J = 1,4
49.    1 2 3                 A(I,J) = 0.
50.  + 1 2 3 4-----<         DO 46020 K = 1,4
51.    1 2 3 4 A--<>          A(I,J) = A(I,J) + B(I,K) * C(K,J)
52.    1 2 3 4--->>> 46020 CONTINUE
```

Rectangular Matrix Multiply

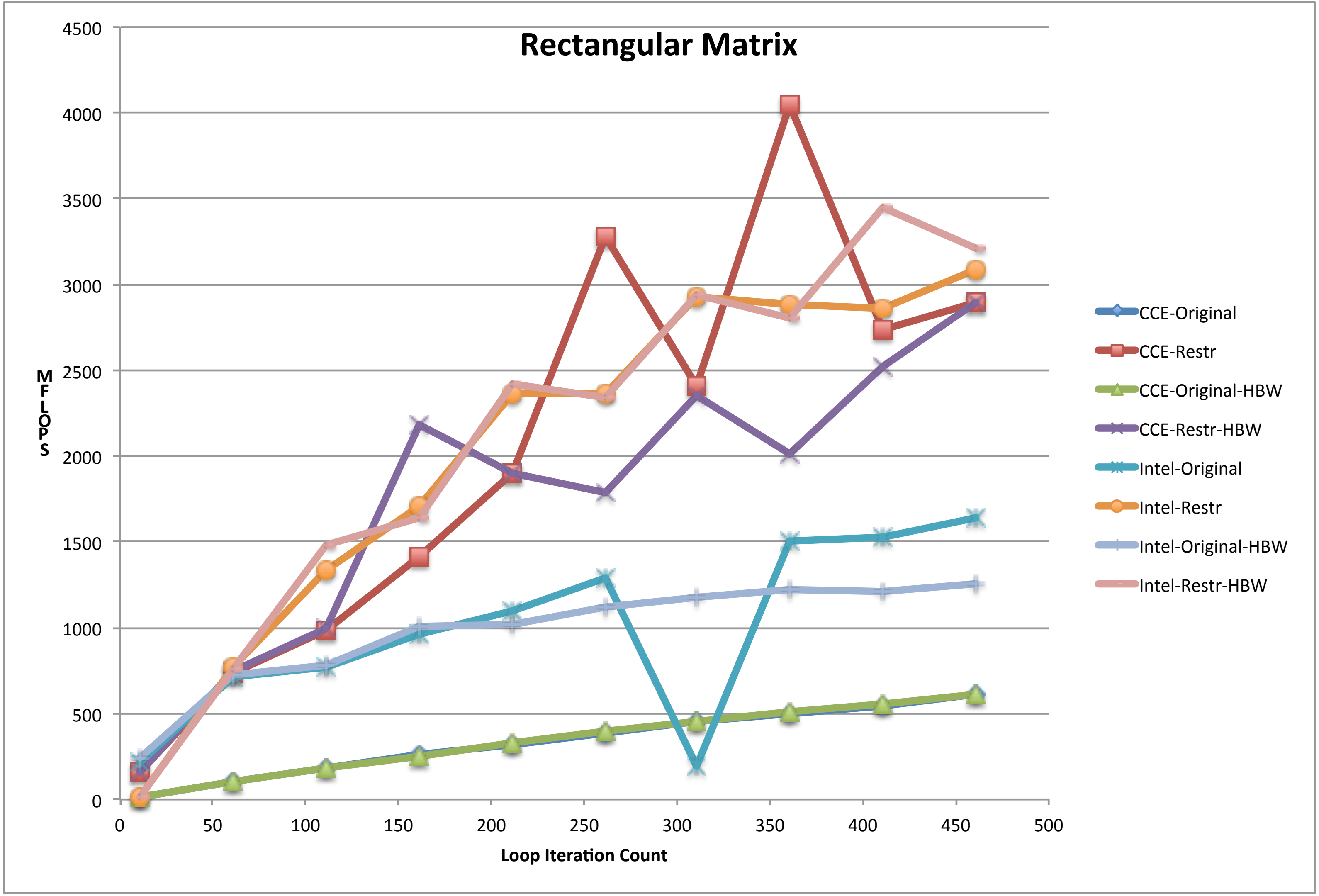
```
69.      1             C       THE RESTRUCTURED
70.      1
71.      1 V---------<        DO 46021 I = 1, N
72.      1 V                    A(I,1) = B(I,1) * C(1,1) + B(I,2) * C(2,1)
73.      1 V                  *          + B(I,3) * C(3,1) + B(I,4) * C(4,1)
74.      1 V                    A(I,2) = B(I,1) * C(1,2) + B(I,2) * C(2,2)
75.      1 V                  *          + B(I,3) * C(3,2) + B(I,4) * C(4,2)
76.      1 V                    A(I,3) = B(I,1) * C(1,3) + B(I,2) * C(2,3)
77.      1 V                  *          + B(I,3) * C(3,3) + B(I,4) * C(4,3)
78.      1 V                    A(I,4) = B(I,1) * C(1,4) + B(I,2) * C(2,4)
79.      1 V                  *          + B(I,3) * C(3,4) + B(I,4) * C(4,4)
80.      1 V---------> 46021 CONTINUE
```

Rectangular Matrix Multiply

```
 LOOP BEGIN at lp46020.f(47,10)
      remark #15541: outer loop was not auto-vectorized: consider using SIMD directive   [ lp46020.f(50,12) ]
      remark #25456: Number of Array Refs Scalar Replaced In Loop: 16

      LOOP BEGIN at lp46020.f(48,11)
         remark #15541: outer loop was not auto-vectorized: consider using SIMD directive   [ lp46020.f(50,12) ]
         remark #25436: completely unrolled by 4

         LOOP BEGIN at lp46020.f(50,12)
            remark #25085: Preprocess Loopnests: Moving Out Load and Store    [ lp46020.f(51,10) ]
            remark #15335: loop was not vectorized: vectorization possible but seems inefficient. Use vector
always directive or -vec-threshold0 to override
            remark #25436: completely unrolled by 4
         LOOP END

         LOOP BEGIN at lp46020.f(50,12)
         LOOP END

         LOOP BEGIN at lp46020.f(50,12)
         LOOP END

         LOOP BEGIN at lp46020.f(50,12)
         LOOP END
      LOOP END
   LOOP END
```

Rectangular Matrix Multiply

```
LOOP BEGIN at lp46020.f(71,10)
  <Peeled loop for vectorization>
     remark #15301: PEEL LOOP WAS VECTORIZED
  LOOP END

  LOOP BEGIN at lp46020.f(71,10)
     remark #15300: LOOP WAS VECTORIZED
     remark #25456: Number of Array Refs Scalar Replaced In Loop: 12
  LOOP END

  LOOP BEGIN at lp46020.f(71,10)
  <Remainder loop for vectorization>
     remark #15301: REMAINDER LOOP WAS VECTORIZED
  LOOP END
```


Rectangular Matrix
MFLOPS
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Optimized Matrix Multiply

```
42.    1               C      THE ORIGINAL
  43.    1
  44.  + 1 2-----------<       DO 46030 J  = 1, N
  45.    1 2 A---------<        DO 46030 I = 1, N
  46.    1 2 A                   A(I,J) = 0.
  47.    1 2 A-------->> 46030 CONTINUE
  48.    1
  49.  + 1 2-----------<       DO 46031   K = 1, N
  50.  + 1 2 3---------<        DO 46031  J = 1, N
  51.  + 1 2 3 4-------<         DO 46031 I = 1, N
  52.    1 2 3 4 A----<>          A(I,J) = A(I,J) + B(I,K) * C(K,J)
  53.    1 2 3 4----->>> 46031 CONTINUE
```

Optimized Matrix Multiply

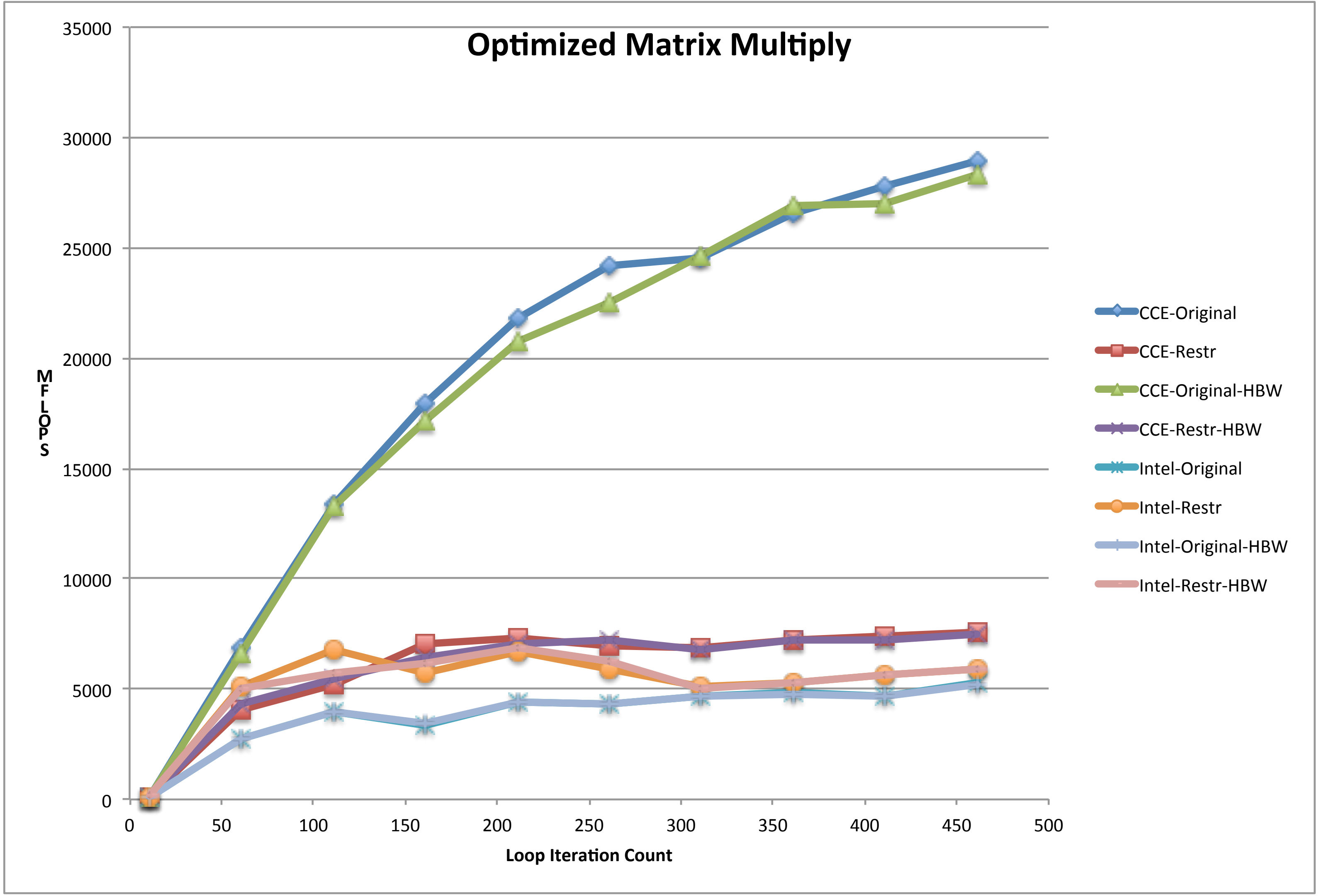
```
70.    1               C       THE RESTRUCTURED
71.    1
72.  + 1 2-----------<       DO 46032  J = 1, N
73.    1 2 A---------<        DO 46032 I = 1, N
74.    1 2 A                   A(I,J)=0.
75.    1 2 A-------->> 46032 CONTINUE
76.    1               C
77.  + 1 2-----------<       DO 46033   K = 1, N-5, 6
78.  + 1 2 r4--------<        DO 46033  J = 1, N
79.    1 2 r4 Vr2----<         DO 46033 I = 1, N
80.    1 2 r4 Vr2               A(I,J) = A(I,J) + B(I,K   ) * C(K   ,J)
81.    1 2 r4 Vr2          *                   + B(I,K+1) * C(K+1,J)
82.    1 2 r4 Vr2          *                   + B(I,K+2) * C(K+2,J)
83.    1 2 r4 Vr2          *                   + B(I,K+3) * C(K+3,J)
84.    1 2 r4 Vr2          *                   + B(I,K+4) * C(K+4,J)
85.    1 2 r4 Vr2          *                   + B(I,K+5) * C(K+5,J)
86.    1 2 r4 Vr2-->>> 46033 CONTINUE
87.    1               C
88.  + 1 2-----------<       DO 46034  KK = K, N
89.  + 1 2 3---------<        DO 46034  J = 1, N
90.  + 1 2 3 4-------<         DO 46034 I = 1, N
91.    1 2 3 4 A----<>          A(I,J) = A(I,J) + B(I,KK) * C(KK ,J)
92.    1 2 3 4----->>> 46034 CONTINUE
```

Optimized Matrix Multiply

```
LOOP BEGIN at lp46030.f(30,10)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at lp46030.f(44,10)
      remark #15542: loop was not vectorized: inner loop was already vectorized

      LOOP BEGIN at lp46030.f(45,11)
      <Peeled loop for vectorization>
         remark #15335: peel loop was not vectorized: vectorization possible but seems inefficient. Use vector
always directive or -vec-threshold0 to override
      LOOP END

      LOOP BEGIN at lp46030.f(45,11)
         remark #15300: LOOP WAS VECTORIZED
      LOOP END

      LOOP BEGIN at lp46030.f(45,11)
      <Remainder loop for vectorization>
         remark #15301: REMAINDER LOOP WAS VECTORIZED
      LOOP END
   LOOP END
```

Optimized Matrix Multiply

```
LOOP BEGIN at lp46030.f(72,10)
     remark #15542: loop was not vectorized: inner loop was already vectorized

     LOOP BEGIN at lp46030.f(73,11)
     <Peeled loop for vectorization>
        remark #15335: peel loop was not vectorized: vectorization possible but seems inefficient.
                               Use vector always directive or -vec-threshold0 to override
     LOOP END

     LOOP BEGIN at lp46030.f(73,11)
        remark #15300: LOOP WAS VECTORIZED
     LOOP END

     LOOP BEGIN at lp46030.f(73,11)
     <Remainder loop for vectorization>
        remark #15301: REMAINDER LOOP WAS VECTORIZED
     LOOP END
  LOOP END

  LOOP BEGIN at lp46030.f(77,10)
     remark #15542: loop was not vectorized: inner loop was already vectorized

     LOOP BEGIN at lp46030.f(78,11)
        remark #15542: loop was not vectorized: inner loop was already vectorized

        LOOP BEGIN at lp46030.f(79,12)
           remark #15300: LOOP WAS VECTORIZED
        LOOP END

        LOOP BEGIN at lp46030.f(79,12)
        <Remainder loop for vectorization>
           remark #15301: REMAINDER LOOP WAS VECTORIZED
        LOOP END
     LOOP END
  LOOP END
```


Optimized Matrix Multiply
MFLOPS
35000
30000
25000
20000
15000
10000
5000
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

IF tests on Loop Indices

```
53.    1               C      THE ORIGINAL
  54.    1
  55.  + 1 i-----------<       DO 47020   J = 1, JMAX
  56.  + 1 i i---------<        DO 47020  K = 1, KMAX
  57.  + 1 i i V-------<         DO 47020 I = 1, IMAX
  58.    1 i i V                  JP         = J + 1
  59.    1 i i V                  JR         = J - 1
  60.    1 i i V                  KP         = K + 1
  61.    1 i i V                  KR         = K - 1
  62.    1 i i V                  IP         = I + 1
  63.    1 i i V                  IR         = I - 1
  64.    1 i i V                  IF (J .EQ. 1)     GO TO 50
  65.    1 i i V                   IF( J .EQ. JMAX) GO TO 51
  66.    1 i i V                    XJ = ( A(I,JP,K) - A(I,JR,K) ) * DA2
  67.    1 i i V                    YJ = ( B(I,JP,K) - B(I,JR,K) ) * DA2
  68.    1 i i V                    ZJ = ( C(I,JP,K) - C(I,JR,K) ) * DA2
  69.    1 i i V                    GO TO 70
  70.    1 i i V            50    J1 = J + 1
  71.    1 i i V                  J2 = J + 2
  72.    1 i i V                  XJ = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
  73.    1 i i V                  YJ = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
  74.    1 i i V                  ZJ = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
  75.    1 i i V                  GO TO 70
  76.    1 i i V            51    J1 = J - 1
  77.    1 i i V                  J2 = J - 2
  78.    1 i i V                  XJ = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
  79.    1 i i V                  YJ = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
  80.    1 i i V                  ZJ = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
  81.    1 i i V            70    CONTINUE
  82.    1 i i V                  IF (K .EQ. 1)     GO TO 52
```

IF tests on Loop Indices

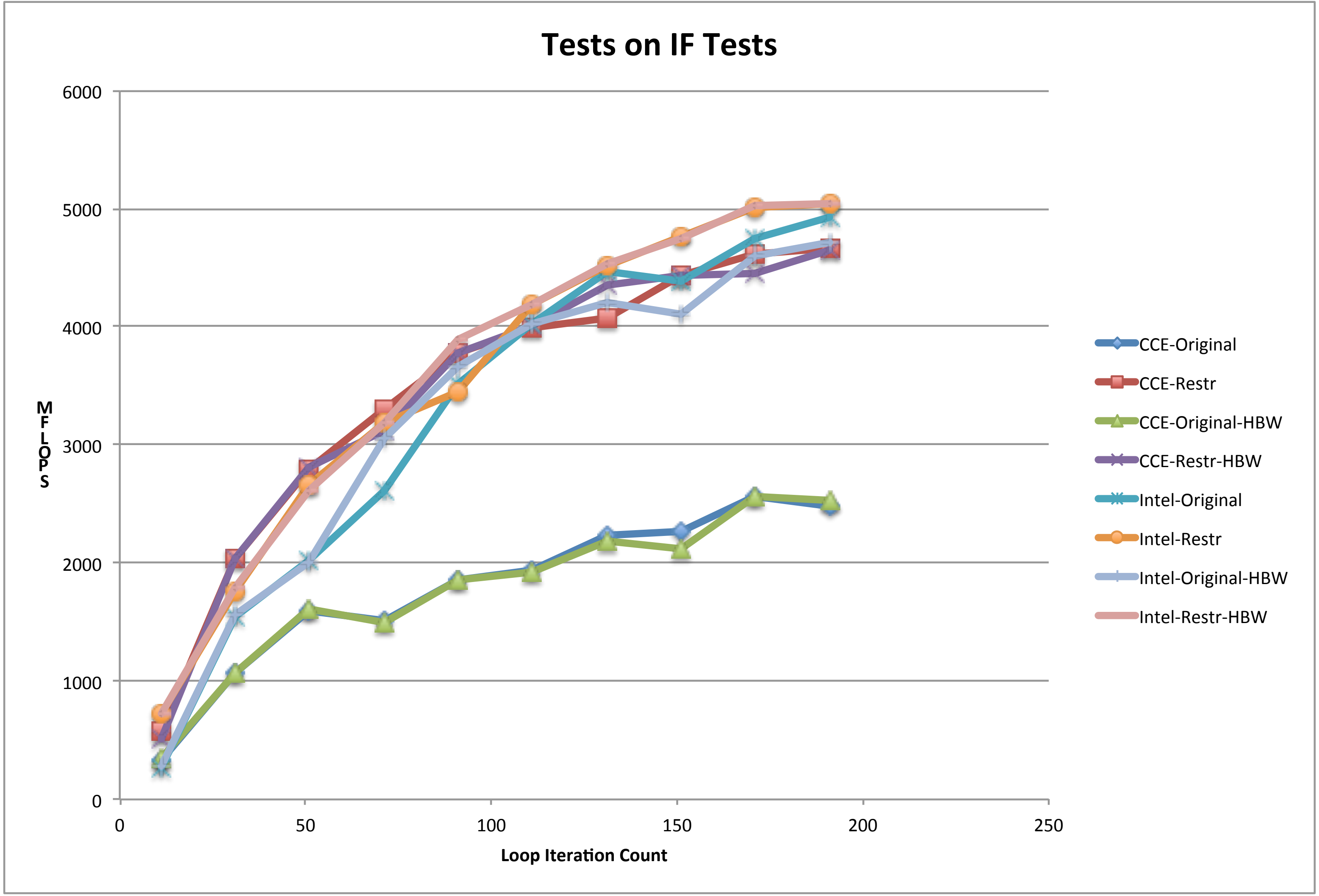
```
 83.    1 i i V                    IF (K .EQ. KMAX) GO TO 53
 84.    1 i i V                     XK = ( A(I,J,KP) - A(I,J,KR) ) * DB2
 85.    1 i i V                     YK = ( B(I,J,KP) - B(I,J,KR) ) * DB2
 86.    1 i i V                     ZK = ( C(I,J,KP) - C(I,J,KR) ) * DB2
 87.    1 i i V                     GO TO 71
 88.    1 i i V           52      K1 = K + 1
 89.    1 i i V                   K2 = K + 2
 90.    1 i i V                   XK = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
 91.    1 i i V                   YK = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
 92.    1 i i V                   ZK = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
 93.    1 i i V                   GO TO 71
 94.    1 i i V           53      K1 = K - 1
 95.    1 i i V                   K2 = K - 2
 96.    1 i i V                   XK = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
 97.    1 i i V                   YK = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
 98.    1 i i V                   ZK = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
 99.    1 i i V           71      CONTINUE
100.    1 i i V                   IF (I .EQ. 1)     GO TO 54
101.    1 i i V                    IF (I .EQ. IMAX) GO TO 55
102.    1 i i V                     XI = ( A(IP,J,K) - A(IR,J,K) ) * DC2
103.    1 i i V                     YI = ( B(IP,J,K) - B(IR,J,K) ) * DC2
104.    1 i i V                     ZI = ( C(IP,J,K) - C(IR,J,K) ) * DC2
105.    1 i i V                     GO TO 60
106.    1 i i V           54      I1 = I + 1
107.    1 i i V                   I2 = I + 2
108.    1 i i V                   XI = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
109.    1 i i V                   YI = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
110.    1 i i V                   ZI = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
111.    1 i i V                   GO TO 60
112.    1 i i V           55      I1 = I - 1
113.    1 i i V                   I2 = I - 2
114.    1 i i V                   XI = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
115.    1 i i V                   YI = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
116.    1 i i V                   ZI = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
117.    1 i i V           60      CONTINUE
118.    1 i i V                   DINV     = XJ * YK * ZI  +  YJ * ZK * XI  +  ZJ * XK * YI
119.    1 i i V              *             - XJ * ZK * YI  -  YJ * XK * ZI  -  ZJ * YK * XI
120.    1 i i V                   D(I,J,K) = 1. / (DINV + 1.E-20)
121.    1 i i V----->>> 47020 CONTINUE
```

IF tests on Loop Indices

```
142.     1                C      THE RESTRUCTURED
143.     1
144.   + 1 2-----------<        DO 47029 J = 1, JMAX
145.   + 1 2 3---------<         DO 47029 K = 1, KMAX
146.     1 2 3
147.     1 2 3                    IF(J.EQ.1)THEN
148.     1 2 3
149.     1 2 3                    J1          = 2
150.     1 2 3                    J2          = 3
151.     1 2 3 Vr2-----<          DO 47021 I = 1, IMAX
152.     1 2 3 Vr2                 VAJ(I) = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
153.     1 2 3 Vr2                 VBJ(I) = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
154.     1 2 3 Vr2                 VCJ(I) = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
155.     1 2 3 Vr2-----> 47021    CONTINUE
156.     1 2 3
157.     1 2 3                    ELSE IF(J.NE.JMAX) THEN
158.     1 2 3
159.     1 2 3                    JP          = J+1
160.     1 2 3                    JR          = J-1
161.     1 2 3 Vr2-----<          DO 47022 I = 1, IMAX
162.     1 2 3 Vr2                 VAJ(I) = ( A(I,JP,K) - A(I,JR,K) ) * DA2
163.     1 2 3 Vr2                 VBJ(I) = ( B(I,JP,K) - B(I,JR,K) ) * DA2
164.     1 2 3 Vr2                 VCJ(I) = ( C(I,JP,K) - C(I,JR,K) ) * DA2
165.     1 2 3 Vr2-----> 47022    CONTINUE
166.     1 2 3
167.     1 2 3                    ELSE
168.     1 2 3
169.     1 2 3                    J1          = JMAX-1
170.     1 2 3                    J2          = JMAX-2
171.     1 2 3 Vr2-----<          DO 47023 I = 1, IMAX
172.     1 2 3 Vr2                 VAJ(I) = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
173.     1 2 3 Vr2                 VBJ(I) = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
174.     1 2 3 Vr2                 VCJ(I) = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
175.     1 2 3 Vr2-----> 47023    CONTINUE
176.     1 2 3
177.     1 2 3                    ENDIF
178.     1 2 3
179.     1 2 3                    IF(K.EQ.1) THEN
180.     1 2 3
181.     1 2 3                    K1          = 2
182.     1 2 3                    K2          = 3
183.     1 2 3 Vr2-----<          DO 47024 I = 1, IMAX
184.     1 2 3 Vr2                 VAK(I) = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
185.     1 2 3 Vr2                 VBK(I) = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
186.     1 2 3 Vr2                 VCK(I) = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
187.     1 2 3 Vr2-----> 47024    CONTINUE
```

IF tests on Loop Indices

```
188.    1 2 3
189.    1 2 3                   ELSE IF(K.NE.KMAX)THEN
190.    1 2 3
191.    1 2 3                   KP          = K + 1
192.    1 2 3                   KR          = K - 1
193.    1 2 3 Vr2-----<         DO 47025 I = 1, IMAX
194.    1 2 3 Vr2                VAK(I) = ( A(I,J,KP) - A(I,J,KR) ) * DB2
195.    1 2 3 Vr2                VBK(I) = ( B(I,J,KP) - B(I,J,KR) ) * DB2
196.    1 2 3 Vr2                VCK(I) = ( C(I,J,KP) - C(I,J,KR) ) * DB2
197.    1 2 3 Vr2-----> 47025   CONTINUE
198.    1 2 3
199.    1 2 3                   ELSE
200.    1 2 3
201.    1 2 3                   K1          = KMAX - 1
202.    1 2 3                   K2          = KMAX - 2
203.    1 2 3 Vr2-----<         DO 47026 I = 1, IMAX
204.    1 2 3 Vr2                VAK(I) = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
205.    1 2 3 Vr2                VBK(I) = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
206.    1 2 3 Vr2                VCK(I) = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
207.    1 2 3 Vr2-----> 47026   CONTINUE
208.    1 2 3                   ENDIF
209.    1 2 3
210.    1 2 3                   I = 1
211.    1 2 3                   I1          = 2
212.    1 2 3                   I2          = 3
213.    1 2 3                   VAI(I) = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
214.    1 2 3                   VBI(I) = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
215.    1 2 3                   VCI(I) = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
216.    1 2 3
217.    1 2 3 Vr2-----<         DO 47027 I = 2, IMAX-1
218.    1 2 3 Vr2                IP          = I + 1
219.    1 2 3 Vr2                IR          = I - 1
220.    1 2 3 Vr2                VAI(I) = ( A(IP,J,K) - A(IR,J,K) ) * DC2
221.    1 2 3 Vr2                VBI(I) = ( B(IP,J,K) - B(IR,J,K) ) * DC2
222.    1 2 3 Vr2                VCI(I) = ( C(IP,J,K) - C(IR,J,K) ) * DC2
223.    1 2 3 Vr2-----> 47027   CONTINUE
224.    1 2 3
225.    1 2 3                   I = IMAX
226.    1 2 3                   I1          = IMAX - 1
227.    1 2 3                   I2          = IMAX - 2
228.    1 2 3                   VAI(I) = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
229.    1 2 3                   VBI(I) = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
230.    1 2 3                   VCI(I) = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
231.    1 2 3
232.    1 2 3 Vr2-----<         DO 47028 I = 1, IMAX
233.    1 2 3 Vr2                DINV = VAJ(I) * VBK(I) * VCI(I) + VBJ(I) * VCK(I) * VAI(I)
234.    1 2 3 Vr2            1        + VCJ(I) * VAK(I) * VBI(I) - VAJ(I) * VCK(I) * VBI(I)
235.    1 2 3 Vr2            2        - VBJ(I) * VAK(I) * VCI(I) - VCJ(I) * VBK(I) * VAI(I)
236.    1 2 3 Vr2                D(I,J,K) = 1. / (DINV + 1.E-20)
237.    1 2 3 Vr2-----> 47028   CONTINUE
238.    1 2 3-------->> 47029 CONTINUE
```

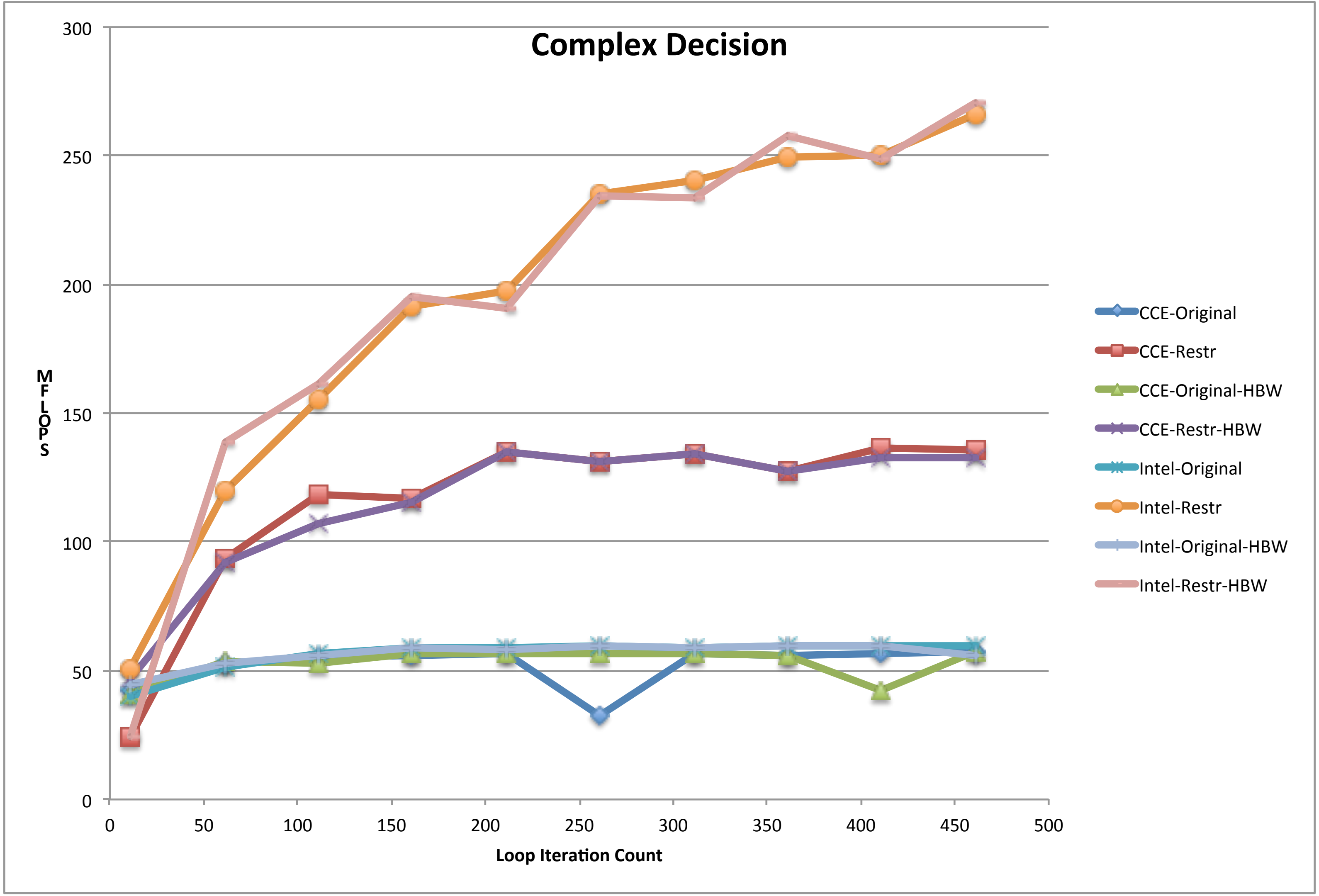

Tests on IF Tests
MFLOPS
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

Complex Decision

```
 80.    1          C      THE ORIGINAL
 81.    1
 82.    1                SUM = 0.0
 83.  + 1 i------<       DO 47080 J = 2, JMAX
 84.  + 1 i i----<       DO 47080 I = 2, N
 85.    1 i i             IF (I .EQ. N) GO TO 47080
 86.    1 i i              IF (A(1,J) .LT. B(1,I)) GO TO 47080
 87.    1 i i               IF (A(1,1) .GT. B(1,I)) GO TO 47080
 88.    1 i i                IF (A(1,J) .GE. B(1,I+1) .AND. I .NE. N)  GO TO 500
 89.    1 i i                 IF (J.EQ.1) GO TO 47080
 90.    1 i i                  IF (A(1,J-1) .LT. B(1,I-1) .AND. I*J .NE. 1)  GO TO 500
 91.    1 i i                   IF (A(1,J-1) .LT. B(1,I)) GO TO 47080
 92.    1 i i        500    CONTINUE
 93.    1 i i               P1    = C(1,I-1)
 94.    1 i i               P2    = D(I-1)
 95.    1 i i               DD    = B(1,I) - B(1,I-1)
 96.    1 i i               P3    = (3.0 * E(I) - 2.0 * P2 - D(I)) / DD
 97.    1 i i               P4    = ( P2 + D(I) - 2.0 * E(I)      ) / DD**2
 98.    1 i i               SUMND = DD * (P1      + DD * (P2 / 2.
 99.    1 i i           *         + DD * (P3 / 3. + DD *  P4 / 4.) ) )
100.    1 i i               SUM   = SUM + SUMND
101.    1 i i--->> 47080 CONTINUE
```

Complex Decision

```
121.    1          C      THE RESTRUCTURED
122.    1
123.    1                SUM = 0.0
124.    1 i------<       DO 47081 J = 2, JMAX
125.    1 i iV---<        DO 47081 I = 2, N-1
126.    1 i iV
127.    1 i iV           LOG1 = A(1,J) .GE. B(1,I)
128.    1 i iV           LOG2 = A(1,1) .LE. B(1,I)
129.    1 i iV           LOG3 = A(1,J) .GE. B(1,I+1)
130.    1 i iV           LOG4 = J .NE. 1
131.    1 i iV           LOG5 = A(1,J-1) .LT. B(1,I-1)
132.    1 i iV           LOG6 = A(1,J-1) .GE. B(1,I)
133.    1 i iV           LOG7 = LOG1 .AND. LOG2 .AND. LOG3 .OR.
134.    1 i iV          *       LOG1 .AND. LOG2 .AND. LOG4 .AND. LOG5 .OR.
135.    1 i iV          *       LOG1 .AND. LOG2 .AND. LOG4 .AND. LOG6
136.    1 i iV            P1    = C(1,I-1)
137.    1 i iV            P2    = D(I-1)
138.    1 i iV            DD    = B(1,I) - B(1,I-1)
139.    1 i iV            IF(.NOT. LOG7) DD=1.0
140.    1 i iV            P3    = (3.0 * E(I) - 2.0 * P2 - D(I)) / DD
141.    1 i iV            P4    = ( P2 + D(I) - 2.0 * E(I)      ) / DD**2
142.    1 i iV            SUMND = 0.0
143.    1 i iV            IF(LOG7) SUMND = DD * (P1      + DD * (P2 / 2.
144.    1 i iV          *                 + DD * (P3 / 3. + DD *  P4 / 4.) ) )
145.    1 i iV            SUM   = SUM + SUMND
146.    1 i iV-->> 47081 CONTINUE
```



Complex Decision
MFLOPS
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
400
450
500
Loop Iteration Count
CCE-Original
CCE-Restr
CCE-Original-HBW
CCE-Restr-HBW
Intel-Original
Intel-Restr
Intel-Original-HBW
Intel-Restr-HBW

VH1 Profile

```
|---------------------------------------------------------------------------
|  69.7% | 275.784732 |        -- |      -- | 9,830,702.0 |USER
||--------------------------------------------------------------------------
||  19.1% |  75.655916 | 2.409537 |   3.1% |   409,600.0 |riemann_
||  10.1% |  39.933326 | 3.687220 |   8.5% | 3,686,400.0 |parabola_
||   7.8% |  30.814069 | 1.117453 |   3.5% |       100.0 |sweepy_
||   6.9% |  27.441332 | 1.908996 |   6.5% |   409,600.0 |remap_
||   5.8% |  22.891882 | 0.595616 |   2.5% |        50.0 |sweepz_
||   3.9% |  15.461957 | 1.012099 |   6.1% |        50.0 |sweepx1_
||   3.5% |  14.017892 | 2.581348 |  15.6% |   819,200.0 |paraset_
||   2.9% |  11.384113 | 1.949496 |  14.6% |   409,600.0 |evolve_
||   2.2% |   8.649484 | 0.464873 |   5.1% |   409,600.0 |ppmlr_
||   2.1% |   8.169585 | 0.321632 |   3.8% |   409,600.0 |states_
||   1.5% |   6.071514 | 0.543860 |   8.2% | 1,228,800.0 |volume_
||   1.3% |   5.015920 | 0.372648 |   6.9% |   409,600.0 |flatten_
||   1.2% |   4.799249 | 0.146185 |   3.0% |        50.0 |sweepx2_
||==========================================================================|==========================
```

Important Loop in Riemann – VH1

```
63.   + 1----< do l = lmin, lmax
64.   + 1 2--<   do n = 1, 12
65.     1 2        pmold(l) = pmid(l)
66.     1 2        wlft (l) = 1.0 + gamfac1*(pmid(l) - plft(l)) * plfti(l)
67.     1 2        wrgh (l) = 1.0 + gamfac1*(pmid(l) - prgh(l)) * prghi(l)
68.     1 2        wlft (l) = clft(l) * sqrt(wlft(l))
69.     1 2        wrgh (l) = crgh(l) * sqrt(wrgh(l))
70.     1 2        zlft (l) = 4.0 * vlft(l) * wlft(l) * wlft(l)
71.     1 2        zrgh (l) = 4.0 * vrgh(l) * wrgh(l) * wrgh(l)
72.     1 2        zlft (l) = -zlft(l) * wlft(l)/(zlft(l) - gamfac2*(pmid(l) - plft(l)))
73.     1 2        zrgh (l) =  zrgh(l) * wrgh(l)/(zrgh(l) - gamfac2*(pmid(l) - prgh(l)))
74.     1 2        umidl(l) = ulft(l) - (pmid(l) - plft(l)) / wlft(l)
75.     1 2        umidr(l) = urgh(l) + (pmid(l) - prgh(l)) / wrgh(l)
76.     1 2        pmid (l) = pmid(l) + (umidr(l) - umidl(l))*(zlft(l) * zrgh(l)) / (zrgh(l
77.     1 2        pmid (l) = max(smallp,pmid(l))
78.     1 2        if (abs(pmid(l)-pmold(l))/pmid(l) < tol ) exit
79.     1 2-->   enddo
80.     1----> enddo
```

Important Loop in Riemann – VH1

```
70.  + 1------< do n = 1, 12
71.  + 1          if(any(not_converge(lmin:lmax)))then
72.    1 Vr2--<   do l = lmin, lmax
73.    1 Vr2        if (not_converge(l))then
74.    1 Vr2        pmold(l)  = pmid(l)
75.    1 Vr2        wlft  (l) = 1.0 + gamfac1*(pmid(l) - plft(l))  * plfti(l)
76.    1 Vr2        wrgh  (l) = 1.0 + gamfac1*(pmid(l) - prgh(l))  * prghi(l)
77.    1 Vr2        wlft  (l) = clft(l)  * sqrt(wlft(l))
78.    1 Vr2        wrgh  (l) = crgh(l)  * sqrt(wrgh(l))
79.    1 Vr2        zlft  (l) = 4.0 * vlft(l)  * wlft(l)  * wlft(l)
80.    1 Vr2        zrgh  (l) = 4.0 * vrgh(l)  * wrgh(l)  * wrgh(l)
81.    1 Vr2        zlft  (l) = -zlft(l)  * wlft(l)/(zlft(l)  - gamfac2*(pmid(l)  - plft(l)))
82.    1 Vr2        zrgh  (l) =  zrgh(l)  * wrgh(l)/(zrgh(l)  - gamfac2*(pmid(l)  - prgh(l)))
83.    1 Vr2        umidl(l)  = ulft(l)  - (pmid(l)  - plft(l))  / wlft(l)
84.    1 Vr2        umidr(l)  = urgh(l)  + (pmid(l)  - prgh(l))  / wrgh(l)
85.    1 Vr2        pmid  (l) = pmid(l)  + (umidr(l)  - umidl(l))*(zlft(l)  * zrgh(l))  / (zrgh(l)  - zlft(l))
86.    1 Vr2        pmid  (l) = max(smallp,pmid(l))
87.    1 Vr2        if  (abs(pmid(l)-pmold(l))/pmid(l)  <  tol  )  then
88.    1 Vr2        not_converge(l)  =  .false.
90.    1 Vr2        endif
91.    1 Vr2        endif
92.    1 Vr2-->   enddo
93.    1            endif
94.    1------> enddo
```

Important Loop in Parabola– VH1

```
35.            !     zero out da(n) if a(n) is a local max/min
36.    f-----<   do n = nmin-1, nmax+1
37.    f           if(diffa(n-1)*diffa(n) < 0.0) da(n) = 0.0
38.    f----->   enddo
39.    V-----<   do n = nmin-1, nmax
40.    V           ar(n) = a(n) + para(n,1)*diffa(n) + para(n,2)*da(n+1) + para(n,3)*da(n)
41.    V           al(n+1) = ar(n)
42.    V----->   enddo
43.    fVr2--<   do n = nmin, nmax
44.    fVr2        onemfl= 1.0 - flat(n)
45.    fVr2        ar(n) = flat(n) * a(n) + onemfl * ar(n)
46.    fVr2        al(n) = flat(n) * a(n) + onemfl * al(n)
47.    fVr2-->   enddo
48.    f-----< do n = nmin, nmax
49.    f         deltaa(n) = ar(n) - al(n)
50.    f         a6(n)     = 6. * (a(n) - .5 * (al(n) + ar(n)))
51.    f         scrch1(n) = (ar(n) - a(n)) * (a(n)-al(n))
52.    f         scrch2(n) = deltaa(n) * deltaa(n)
53.    f         scrch3(n) = deltaa(n) * a6(n)
54.    f-----> enddo
55.    f-----< do n = nmin, nmax
56.    f         if(scrch1(n) <= 0.0) then
57.    f           ar(n) = a(n)
58.    f           al(n) = a(n)
59.    f         endif
60.    f         if(scrch2(n) < +scrch3(n)) al(n) = 3. * a(n) - 2. * ar(n)
61.    f         if(scrch2(n) < -scrch3(n)) ar(n) = 3. * a(n) - 2. * al(n)
62.    f-----> enddo
63.    f-----< do n = nmin, nmax
64.    f         deltaa(n)= ar(n) - al(n)
65.    f         a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
66.    f-----> enddo
```

Important Loop in Parabola– VH1

```

        !     zero out da(n) if a(n) is a local max/min
f-----<   do n = nmin-1, nmax+1
f           if(diffa(n-1)*diffa(n) < 0.0) da(n) = 0.0
f----->   enddo
V-----<   do n = nmin-1, nmax
V           ar(n) = a(n) + para(n,1)*diffa(n) + para(n,2)*da(n+1) + para(n,3)*da(n)
V           al(n+1) = ar(n)
V----->   enddo
Vr2---<   do n = nmin, nmax
Vr2       onemfl= 1.0 - flat(n)
Vr2       ar(n) = flat(n) * a(n) + onemfl * ar(n)
Vr2       al(n) = flat(n) * a(n) + onemfl * al(n)
Vr2       deltaan = ar(n) - al(n)
Vr2       a6n     = 6. * (a(n) - .5 * (al(n) + ar(n)))
Vr2       scrch1n = (ar(n) - a(n)) * (a(n)-al(n))
Vr2       scrch2n = deltaan * deltaan
Vr2       scrch3n = deltaan * a6n
Vr2       if(scrch1n <= 0.0) then
Vr2         ar(n) = a(n)
Vr2         al(n) = a(n)
Vr2       endif
Vr2       if(scrch2n < +scrch3n) al(n) = 3. * a(n) - 2. * ar(n)
Vr2       if(scrch2n < -scrch3n) ar(n) = 3. * a(n) - 2. * al(n)
Vr2       deltaa(n)= ar(n) - al(n)
Vr2       a6(n) = 6. * (a(n) - .5 * (al(n) + ar(n)))
Vr2---> enddo
```

VH1 Profile after optimization

```
|  64.5% | 233.814721 |        -- |     -- | 9,830,702.0 |USER
||-------------------------------------------------------------------------
||  11.3% |  40.996789 |  3.114000 |   7.1% | 3,686,400.0 |parabola_
||   8.7% |  31.436593 |  1.140046 |   3.5% |       100.0 |sweepy_
||   8.1% |  29.199941 | 23.521407 |  44.6% |   409,600.0 |riemann_
||   7.7% |  27.759222 |  1.217341 |   4.2% |   409,600.0 |remap_
||   6.3% |  22.928404 |  0.550848 |   2.3% |        50.0 |sweepz_
||   4.2% |  15.280661 |  1.449079 |   8.7% |        50.0 |sweepx1_
||   4.1% |  14.690055 |  0.674970 |   4.4% |   819,200.0 |paraset_
||   3.3% |  11.803449 |  0.951406 |   7.5% |   409,600.0 |evolve_
||   2.5% |   8.876112 |  0.571191 |   6.0% |   409,600.0 |ppmlr_
||   2.4% |   8.679586 |  0.386993 |   4.3% |   409,600.0 |states_
||   1.8% |   6.487615 |  0.360199 |   5.3% | 1,228,800.0 |volume_
||   1.4% |   5.148014 |  0.201405 |   3.8% |   409,600.0 |flatten_
||   1.3% |   4.748816 |  0.153916 |   3.1% |        50.0 |sweepx2_
||   1.0% |   3.595215 |  0.336065 |   8.6% | 1,228,800.0 |forces_
||=========================================================================
```

Where are we

- **Understand that Memory Bandwidth Utilization is first and foremost**
- **Then we vectorize major computational kernels**
- **Next we need to investigate how best to utilize available cores**
  - First we will look at the best OpenMP examples
  - Then we will look at smaller OpenMP examples
  - Conclude what is important in the OpenMP implementation
  - Lastly look at examples of identifying how many MPI tasks we should run on the node versus how many shared memory threads we should use

SPMD OpenMP

Can we take clues from all-MPI advantages and disadvantages?

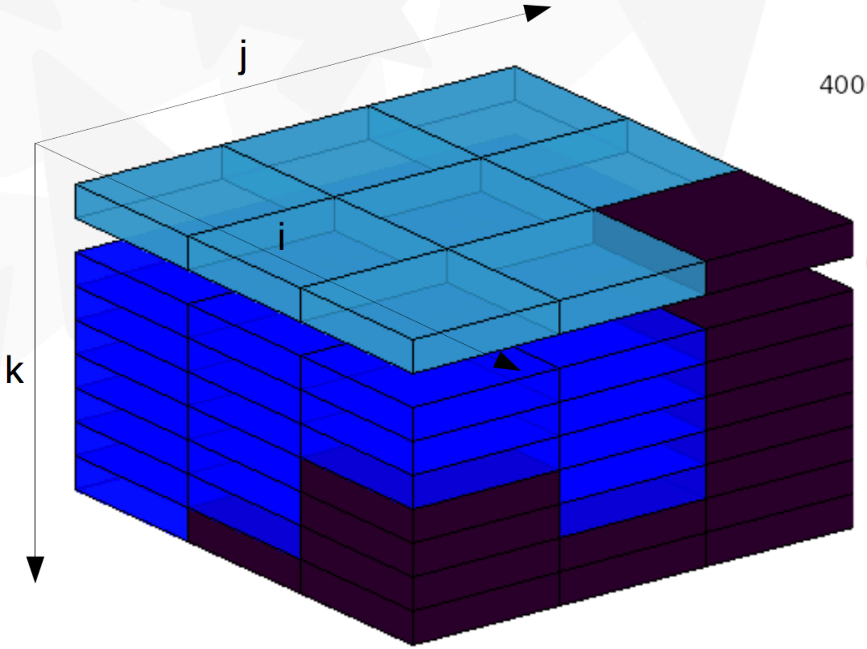
- **Can we force locality like MPI does?**
  - This requires that the thread that uses the data, allocates the data within its NUMA region
  - Tradition OpenMP has no notion of locality
- **Can we allow threads to work asynchronously?**
  - Tradition OpenMP implies barriers after a parallel region
  - Loop level parallelism forces too much synchronization
  - With SPMD OpenMP, user has complete control over synchronization
- **Can we somehow control scheduling of the threads to enable more dynamic re-distribution of work**
  - With SPMD OpenMP, user can write sophistication thread scheduling routine

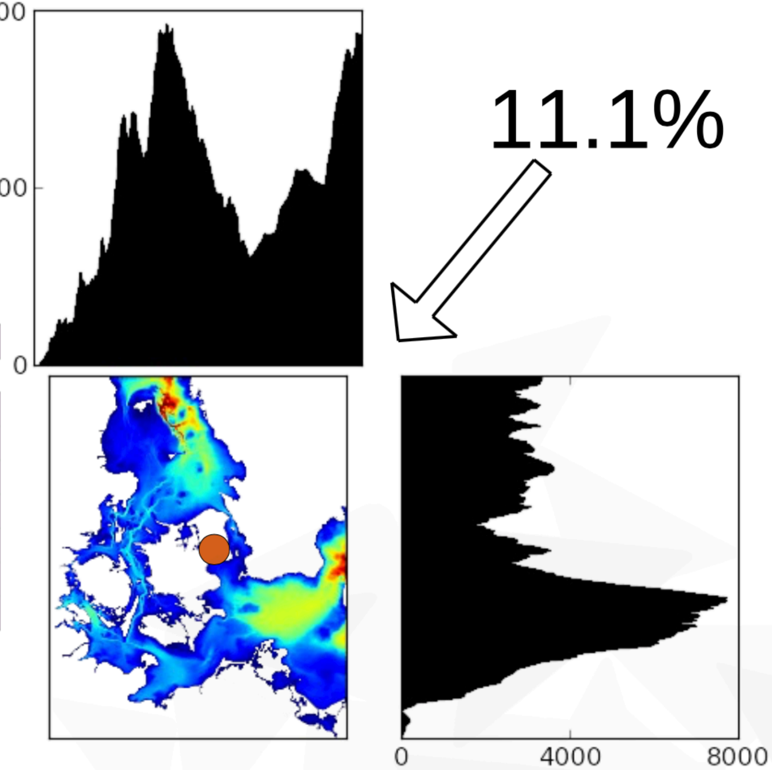
HBM code modernization -

insights from a Xeon Phi experiment

Jacob Weismann Poulsen, DMI, Denmark
Per Berg, DMI, Denmark
Karthik Raman, Intel, USA

The data is sparse and highly irregular

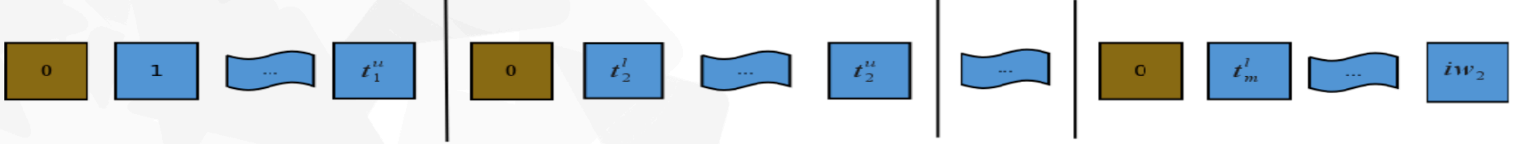




11.1%

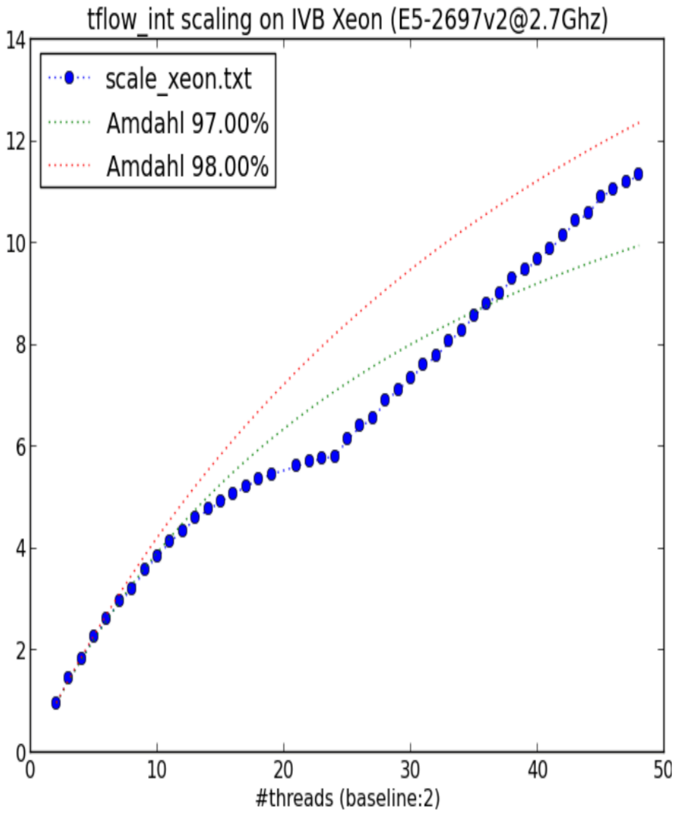
Data layout for threads (or tasks + explicit halo)

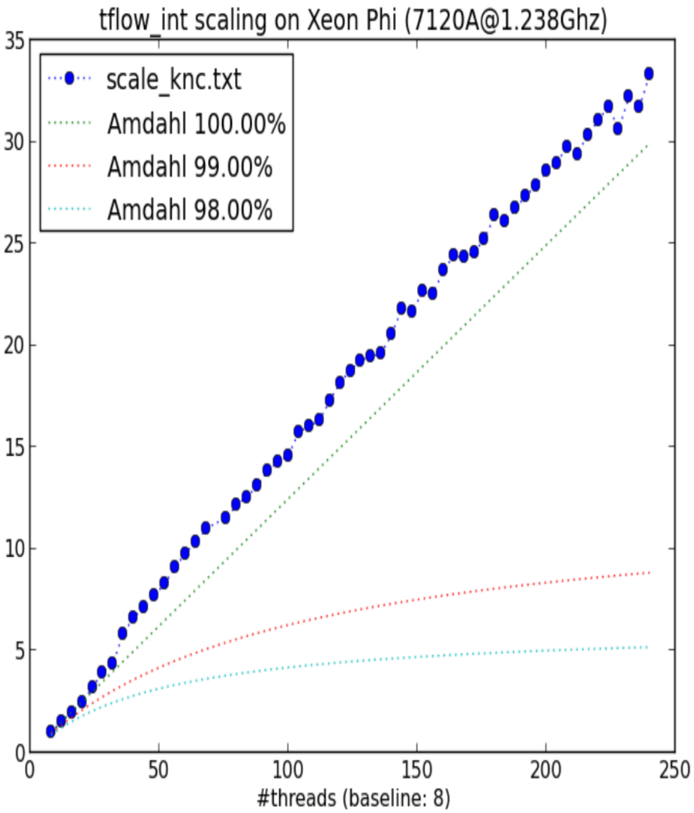
- Each tread will handle a subinterval of columns:



- Another layout of the columns will impose another decomposition for the threads (and the tasks).

```
...
!$OMP PARALLEL DEFAULT(SHARED)
call foo( ... );call bar(...); ...
!$OMP BARRIER
call halo_update(...)
!$OMP BARRIER
call baz( ... );call quux(...); ...
!$OMP END PARALLEL
...
subroutine foo(...)
  ...
  call domp_get_domain(kh, 1, iw2, nl, nu, idx)
  do iw=nl,nu
    i = ind(1,iw)
    j = ind(2,iw)
    ! all threadlocal wet-points (:,:,:) are reached here
    ...
  enddo
end subroutine f
```


tflow_int scaling on IVB Xeon (E5-2697v2@2.7Ghz)
scale_xeon.txt
Amdahl 97.00%
Amdahl 98.00%
0
2
4
6
8
10
12
14
0
10
20
30
40
50
#threads (baseline:2)


tflow_int scaling on Xeon Phi (7120A@1.238Ghz)
scale_knc.txt
Amdahl 100.00%
Amdahl 99.00%
Amdahl 98.00%
0
5
10
15
20
25
30
35
0
50
100
150
200
250
#threads (baseline: 8)

High-level OpenMP and Thread Scalable MPI-RMA:

Application Study with the Wombat Astrophysical MHD Code

Dr. Peter Mendygral
Cray Inc.

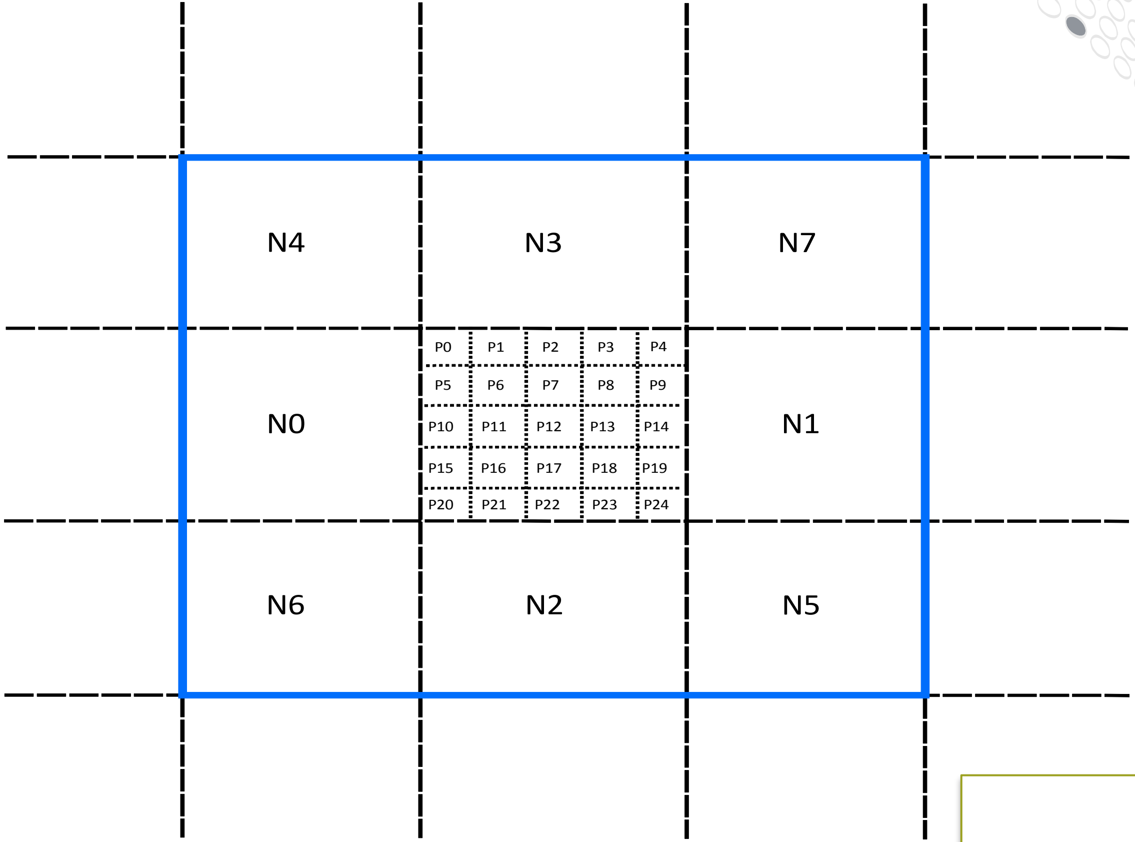
Primary Development Concerns

- **Two main issues drove the design of Wombat and explain why other codes have not been sufficient for the science**
  - Scaling to extreme core count required for resolution requirements
  - Load balancing for SMR/AMR and dark matter particles

- **The approach to these problems in Wombat was**
  - Make communication matter as little as possible
  - Wide OpenMP on a node to soften impacts of load imbalances (and hardware imbalances) as much as possible before communicating work between ranks
  - Data structures that reduce AMR/SMR complexity and avoid significant global communication for refined patch tracking
  - Do it all in Fortran as that's what works best for me
    - Wombat uses object oriented features from Fortran 2003 and 2008

Domain Decomposition

- Domain decomposition is represented in the data classes and structures
- “Domain” is an array of “Patches” managed by a MPI rank
- “Patch” is a self-contained, self-describing piece of the world grid at some fixed logical location
- Tunable neighborhood (blue box) sets the horizon of ranks that can be communicated with



Domains and Patch

- **Domains manage bookkeeping for an array of Patches**
  - Track a Patch's neighbors
  - Manage allocating/deallocating a Patch's internal grid arrays as needed
  - Multiple Domains are used on a rank for accepting Patches from neighbors

- **Patches are**
  - Of some uniform fixed size (for a given Domain refinement level)
  - Fixed in a location that is known for all times by all ranks

- **Patches provide**
  - An atomic unit of work
  - Units to thread across
  - Unit to transfer for load balancing
  - A level of cache blocking

High-level OpenMP

- Two choices (could be used at the same time) for threading the Domain/Patch design

Option A

```
! Move OpenMP near the top of the call stack

!#OMP PARALLEL
DO WHILE (t .LT. tend)

   !#OMP DO SCHEDULE(GUIDED)
   DO patch = 1, npatches

      CALL update_patch()

      ! All threads drive MPI

   END DO

END DO
```

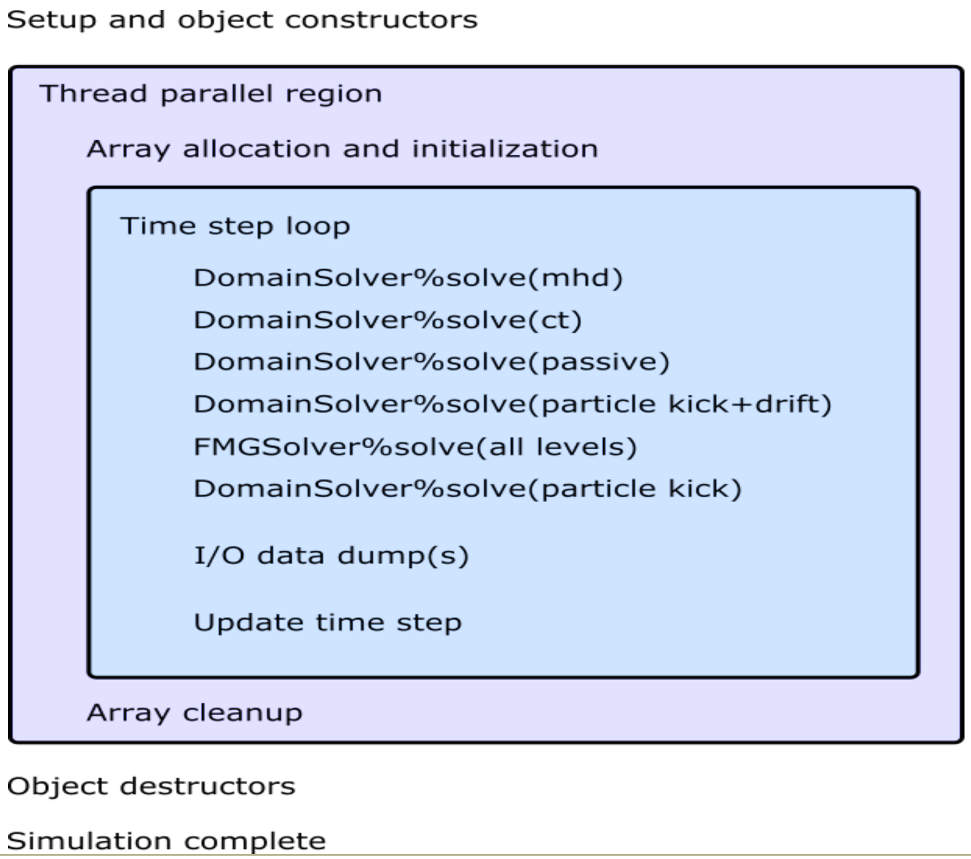
Option B

```
! Keep OpenMP within a “compute” loop

DO WHILE (t .LT. tend)

   DO patch = 1, npatches

      CALL update_patch()

      ! MPI driven by single thread

   END DO

END DO

SUBROUTINE update_patch()

   !$OMP PARALLEL DO SCHEDULE(STATIC)
   DO i = 1, nx
   …do work…
   END DO

END SUBROUTINE
```

High-level OpenMP

- **Benefits of OpenMP near the top of the stack**
  - Application more closely mimics completely independent processes
    - Less likely to be in the same portion of code at the same time
    - Bandwidth competition may decrease
    - Amdahl's law
  - Threads are less coupled => infrequent thread synchronization
  - Much less likely to have issues with memory conflicts between threads
  - Simpler to implement when done right
    - Large reduction in the amount of OpenMP directives
    - Very little variable scoping needed as most everything is shared => reduced memory footprint

Wombat Driver and Parallel Region



Setup and object constructors

- Thread parallel region
  - Array allocation and initialization
  - Time step loop
    - DomainSolver%solve(mhd)
    - DomainSolver%solve(ct)
    - DomainSolver%solve(passive)
    - DomainSolver%solve(particle kick+drift)
    - FMGSolver%solve(all levels)
    - DomainSolver%solve(particle kick)
    - I/O data dump(s)
    - Update time step
  - Array cleanup

Object destructors

Simulation complete

Communication Concerns

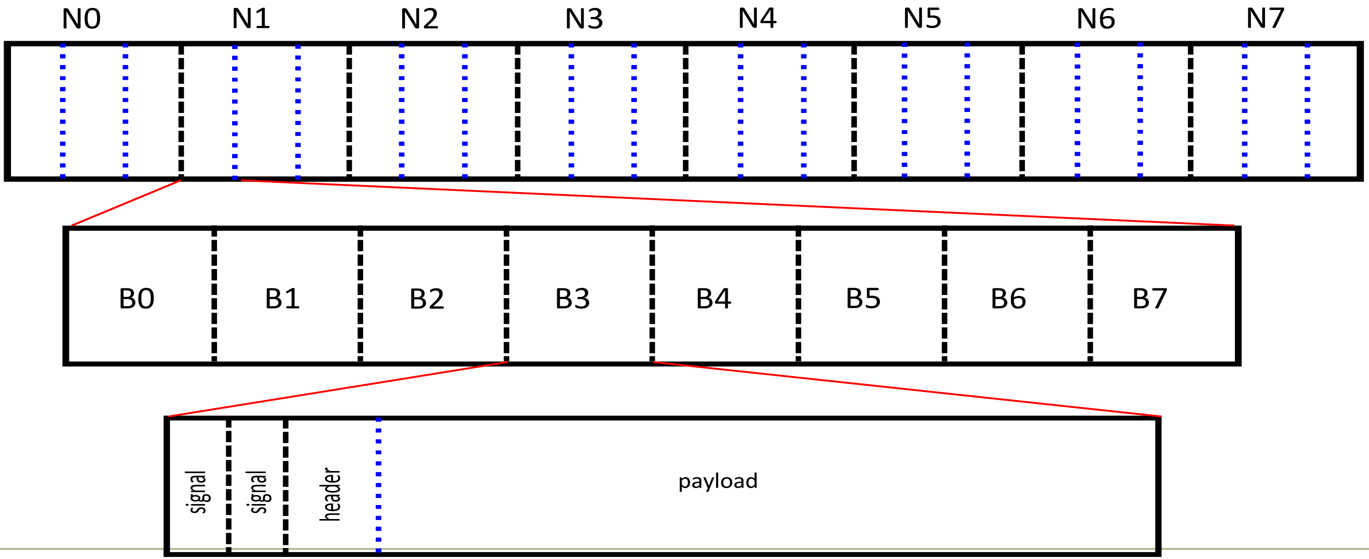
- **If a rank is made much wider with threads, serialization around MPI will limit thread scaling and overall performance**
  - Nearly all MPI libraries implement thread safety with a global lock
  - Cray is addressing this issue
    - Released per-object lock library
    - Threading enhancements under design now for two-sided (released per-object lock library a first step)

- **Wide OpenMP also means more communication to process**
  - Every Patch now has its own smaller boundaries to communicate
  - Starts tipping the behavior towards the message rate limit
  - Two-sided tag matching cannot be done in parallel and will limit thread scaling
    - May start hitting tag limit

- **Slower serial performance of KNL => maybe look for the lightest weight MPI layer available**
  - MPI-RMA over DMAPP on Cray systems is a thin software layer that achi similar performance to SHMEM

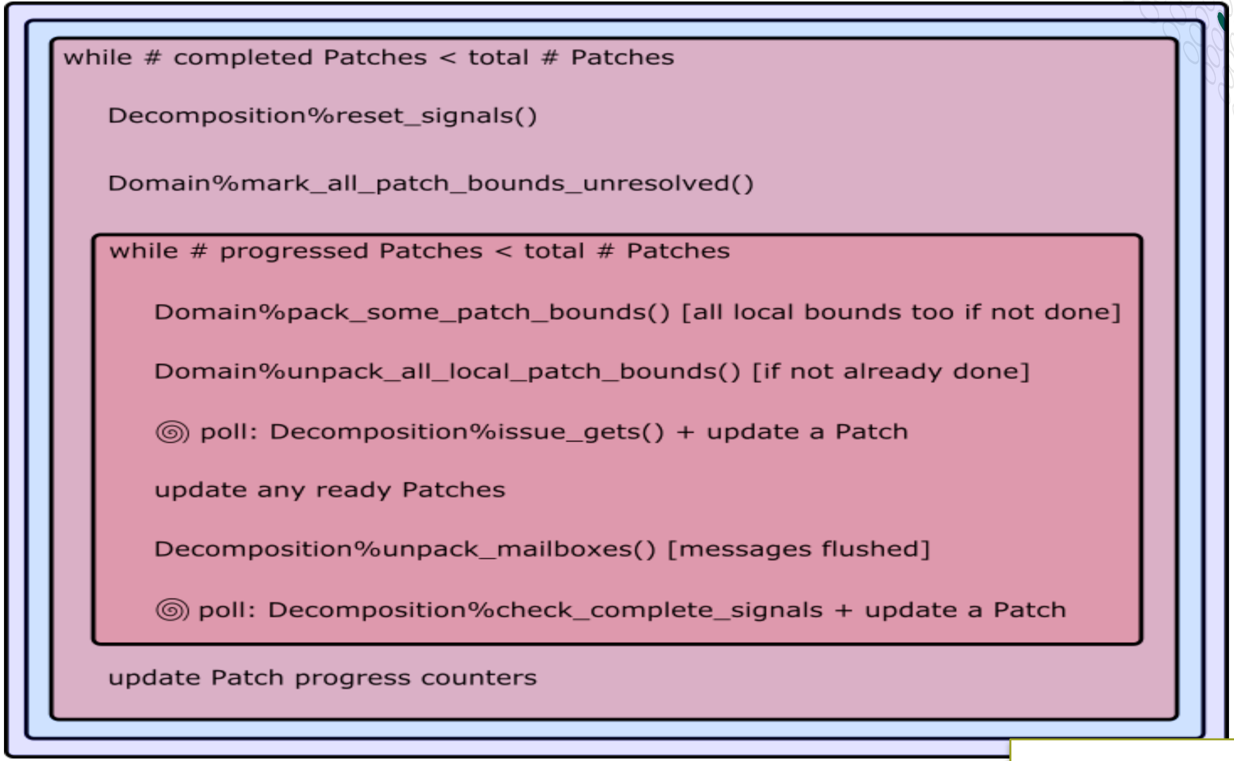
Single RMA Window Buffer

- Single buffer used for (almost) all communication
- Messages can be processed concurrently if MPI allows it
- Design is similar to mailboxes within MPI
  - Can process an arbitrary amount of communication



Generalized Update+Comm Engine

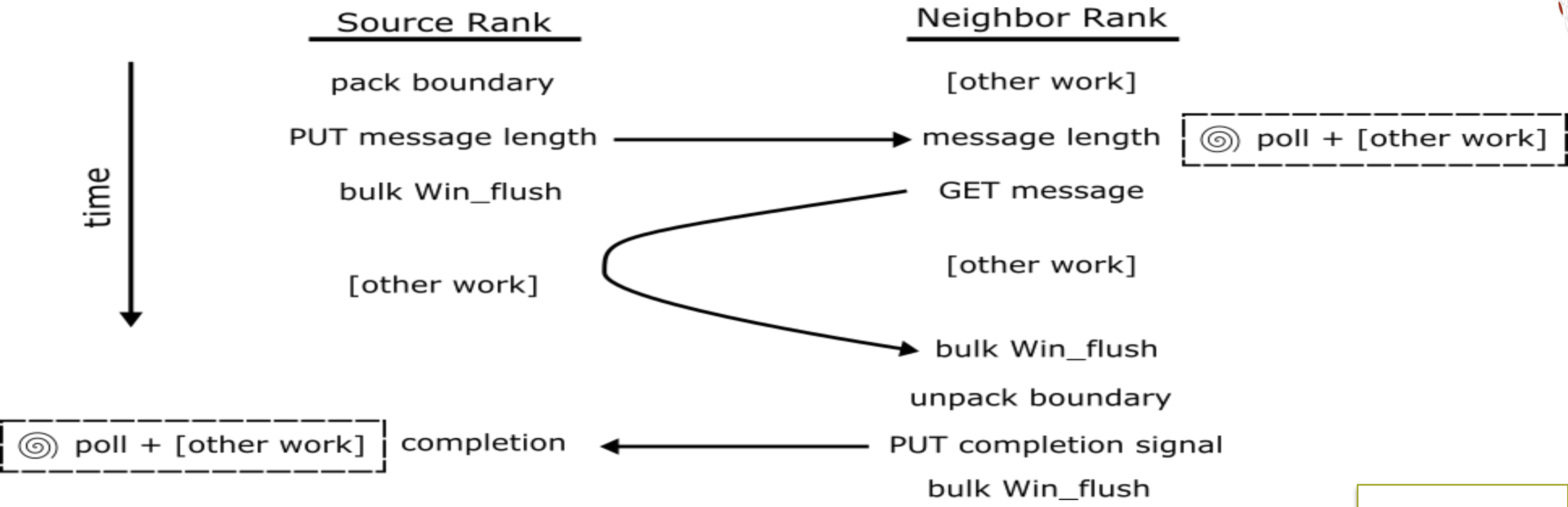
- DomainSolver class
- Single class method capable of driving any of the solvers' data exchange and update process
- Used by every solver in Wombat



```
while # completed Patches < total # Patches
    Decomposition%reset_signals()
    Domain%mark_all_patch_bounds_unresolved()
    while # progressed Patches < total # Patches
        Domain%pack_some_patch_bounds() [all local bounds too if not done]
        Domain%unpack_all_local_patch_bounds() [if not already done]
        ◎ poll: Decomposition%issue_gets() + update a Patch
        update any ready Patches
        Decomposition%unpack_mailboxes() [messages flushed]
        ◎ poll: Decomposition%check_complete_signals + update a Patch
    update Patch progress counters
```

RMA Boundary Communication Cycle

- **Single passive RMA exposure epoch used for the duration of the application**
  - No explicit synchronization between ranks
  - RMA semantics make computation/communication overlap simpler to achieve



time

Source Rank

pack boundary

PUT message length

bulk Win_flush

[other work]

poll + [other work]

completion

Neighbor Rank

[other work]

message length

poll + [other work]

GET message

[other work]

bulk Win_flush

unpack boundary

PUT completion signal

bulk Win_flush

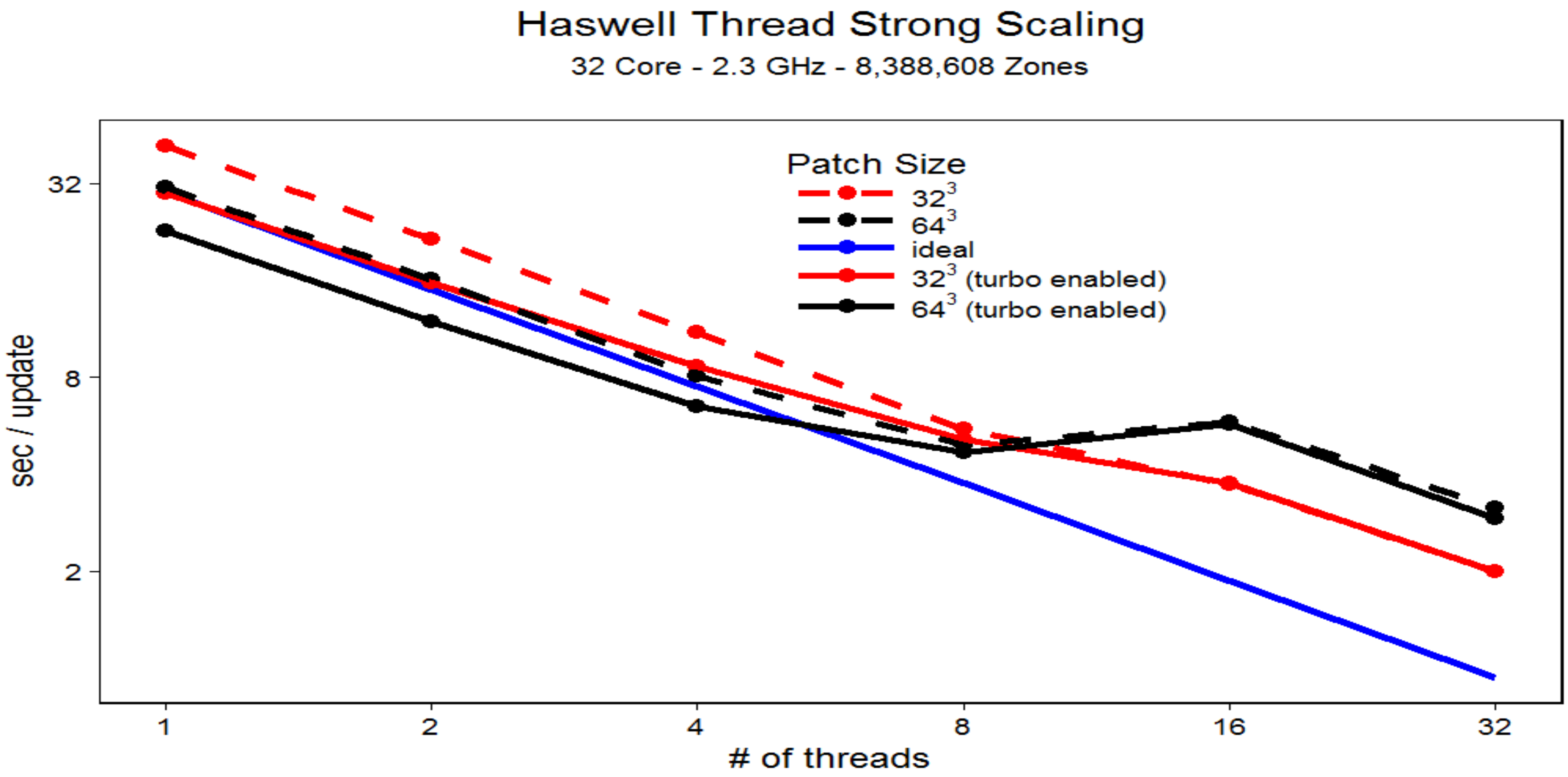
Thread Hot MPI-RMA

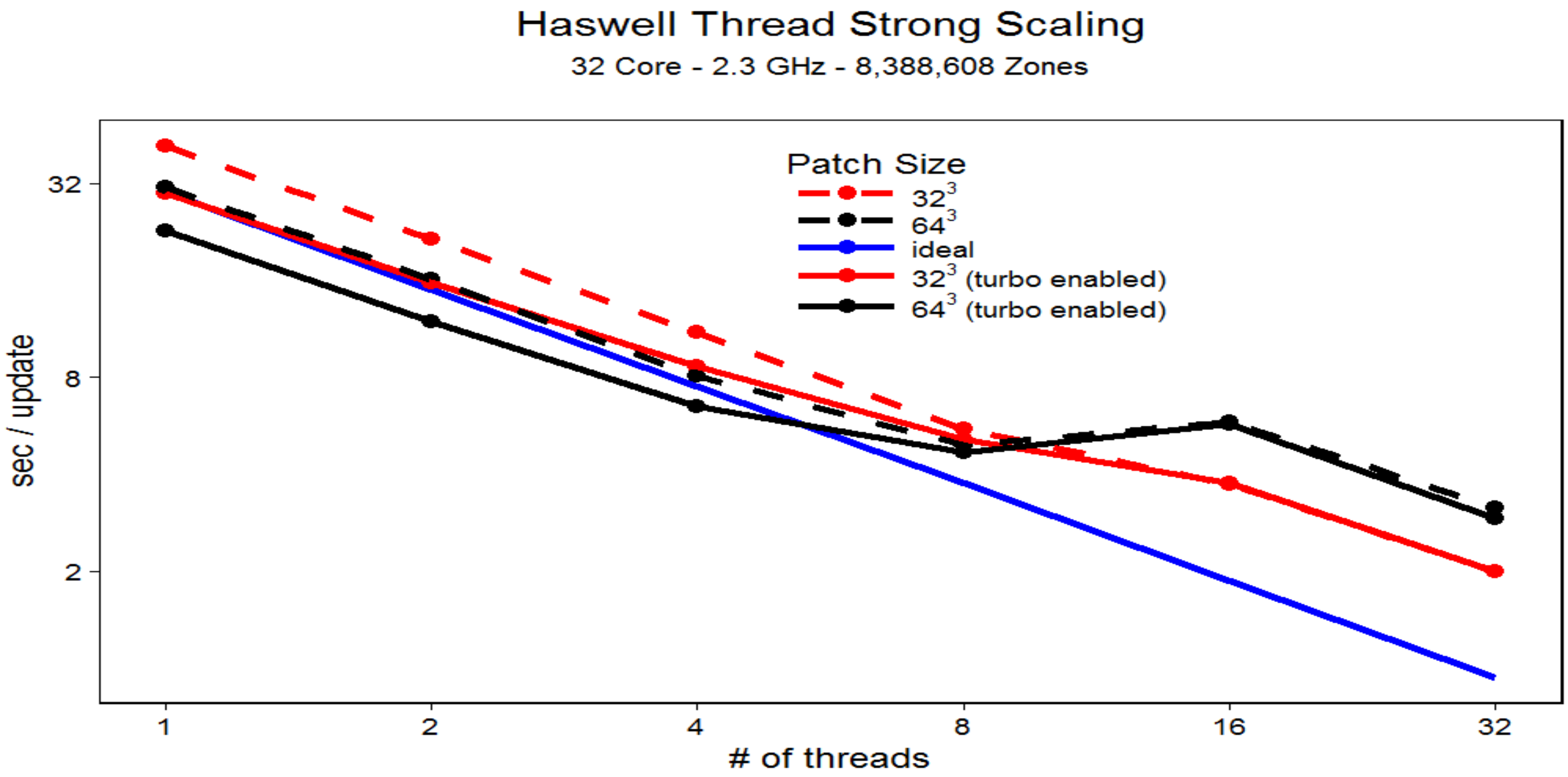
- **DMAPP library was enhanced to be "thread hot" for SHMEM**
  - "thread hot" is more than "thread safe"
  - "thread hot" implies concurrency and performance across threads was central to the design
- **MPI-RMA over DMAPP leverages this feature as of MPT 7.3.2**
  - No locks used in DMAPP layer
  - Very light weight locking in MPI layer
  - Design makes it very likely that locks are uncontended
  - Network resources efficiently managed among threads
  - Performance approaches that of N independent processes when using N threads
- **Example on HSW with 16 threads each on 2 nodes**
  - OSU passive MPI_Put bandwidth for 8 B message
  - MPT 7.3.1 = 5.27 MB/s
  - MPT 7.3.2 = 399.9 MB/s
  - 75X improvement

Wombat Performance

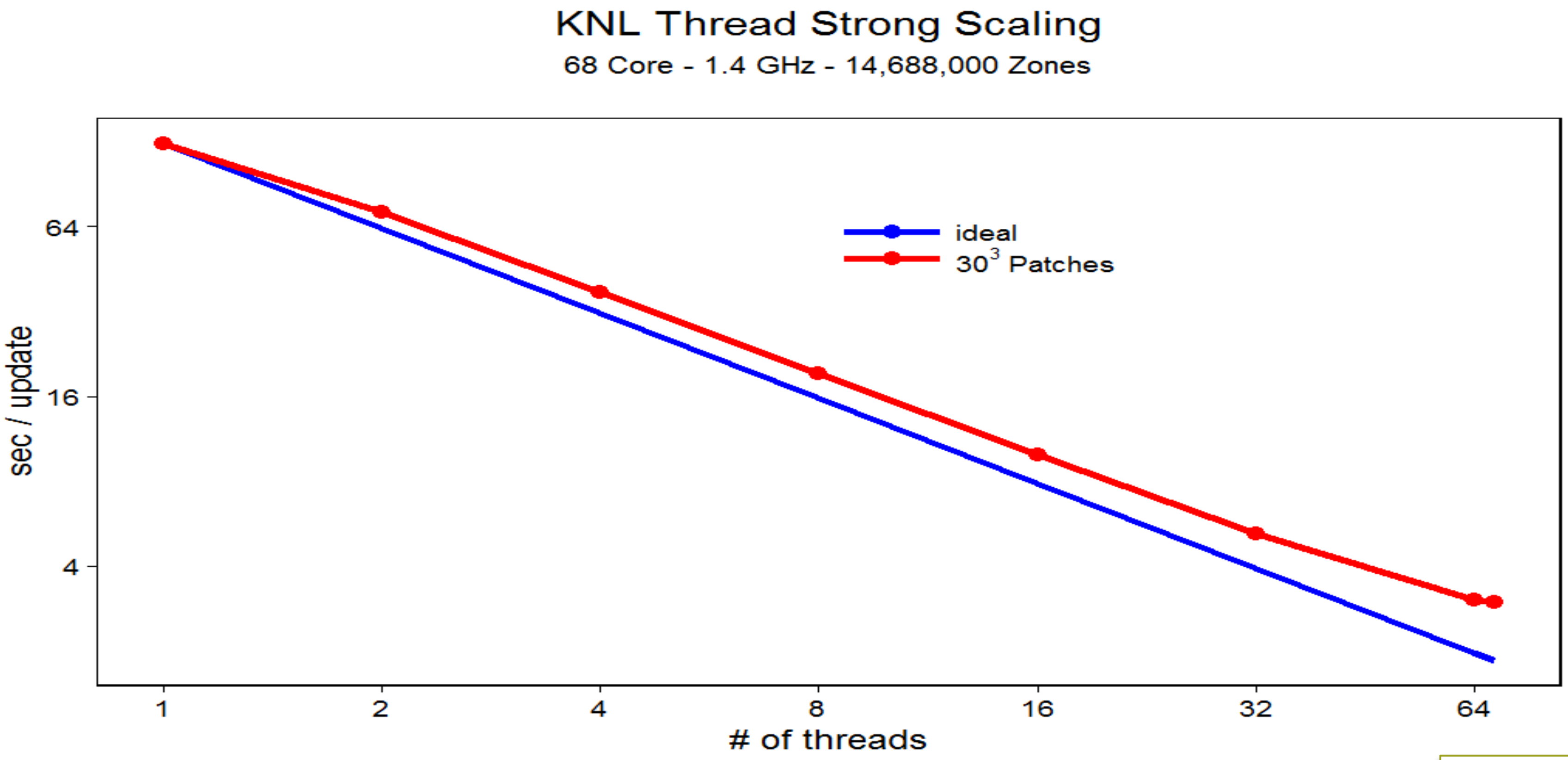
Benchmarking Notes

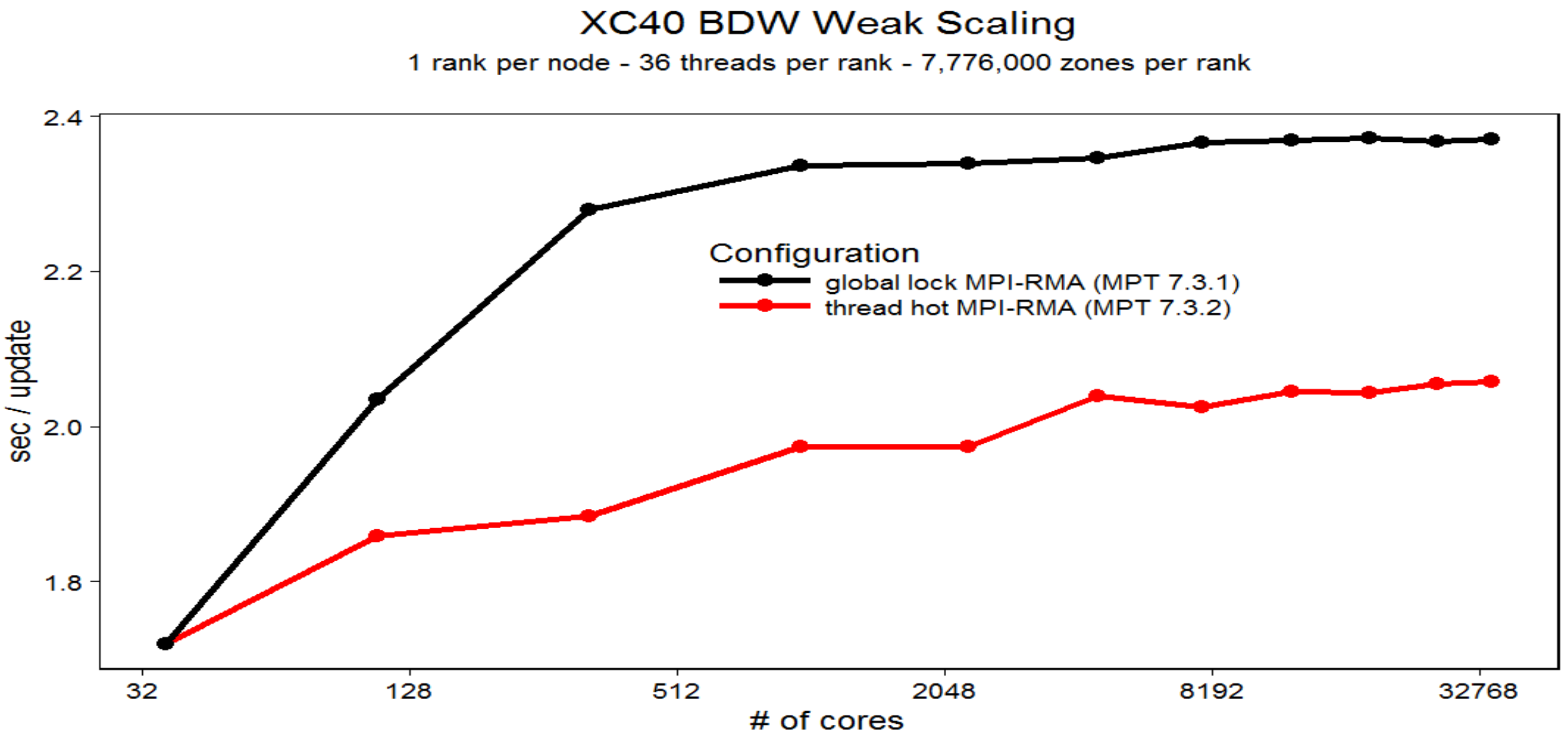
- **Only MHD was enabled on fixed grid running an MHD shock tube test**
- **In all cases 2 MB hugepages were used**
  - module load craype-hugepages2MB
  - Loaded at link and run time
- **The following environment variables were set**
  - export HUGETLB_NO_RESERVE=yes
  - export MPICH_MAX_THREAD_SAFETY=multiple
  - export MPICH_RMA_OVER_DMAPP=1
- **CCE was always used**
  - Intel 15 fails to vectorize key routines (see later slide on vector length sensitivity)
  - Intel 16 hits double free corruption in OpenMP run-time. Investigation on-going.
- **KNL was configured with MCDRAM as cache**


Haswell Thread Strong Scaling
32 Core - 2.3 GHz - 8,388,608 Zones
Patch Size
32^3
64^3
ideal
32^3 (turbo enabled)
64^3 (turbo enabled)
sec / update
32
8
2
1
2
4
8
16
32
# of threads



- **Tunable Patch size very important to performance**

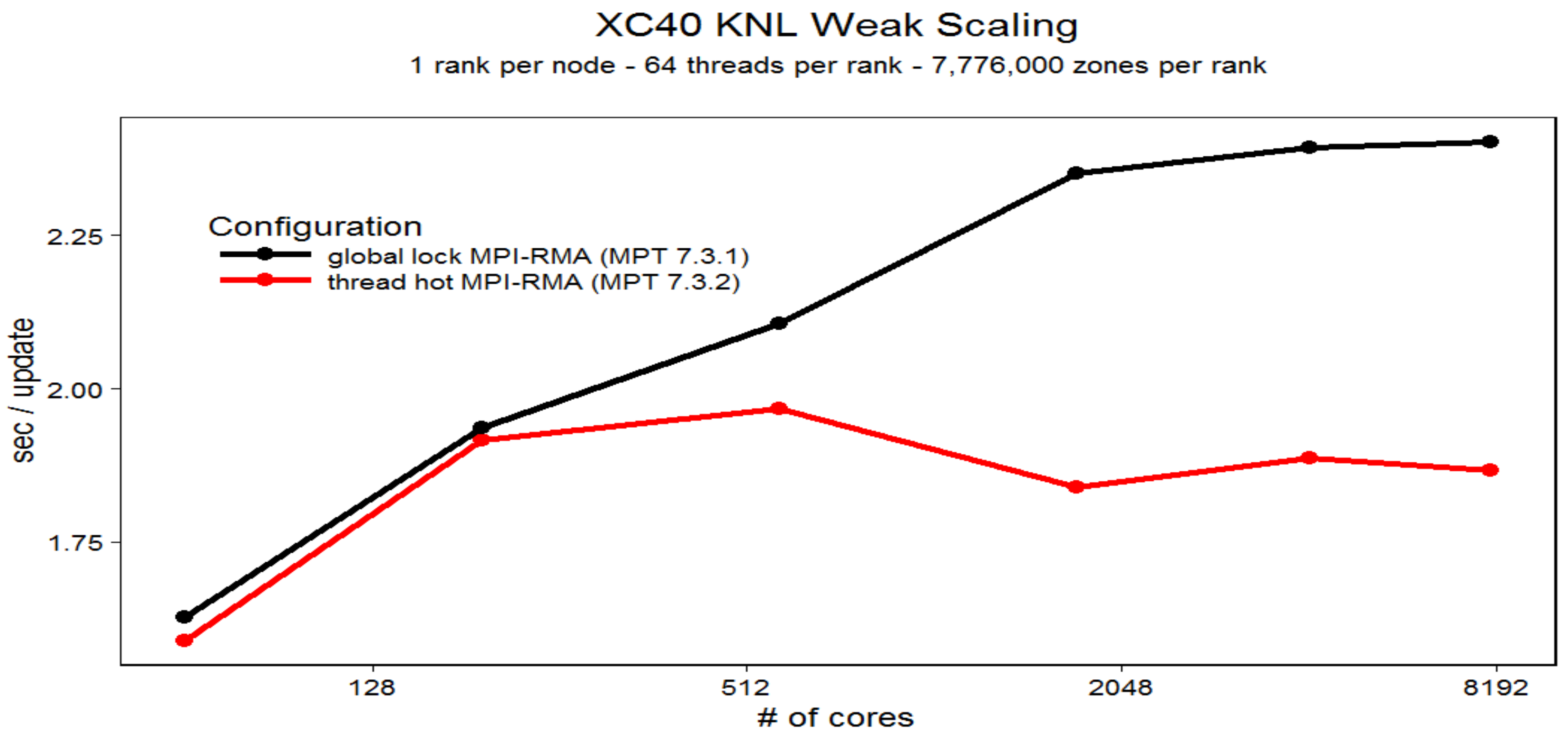

KNL Thread Strong Scaling
68 Core - 1.4 GHz - 14,688,000 Zones
sec / update
64
16
4
1
2
4
8
16
32
64
# of threads
ideal
30³ Patches

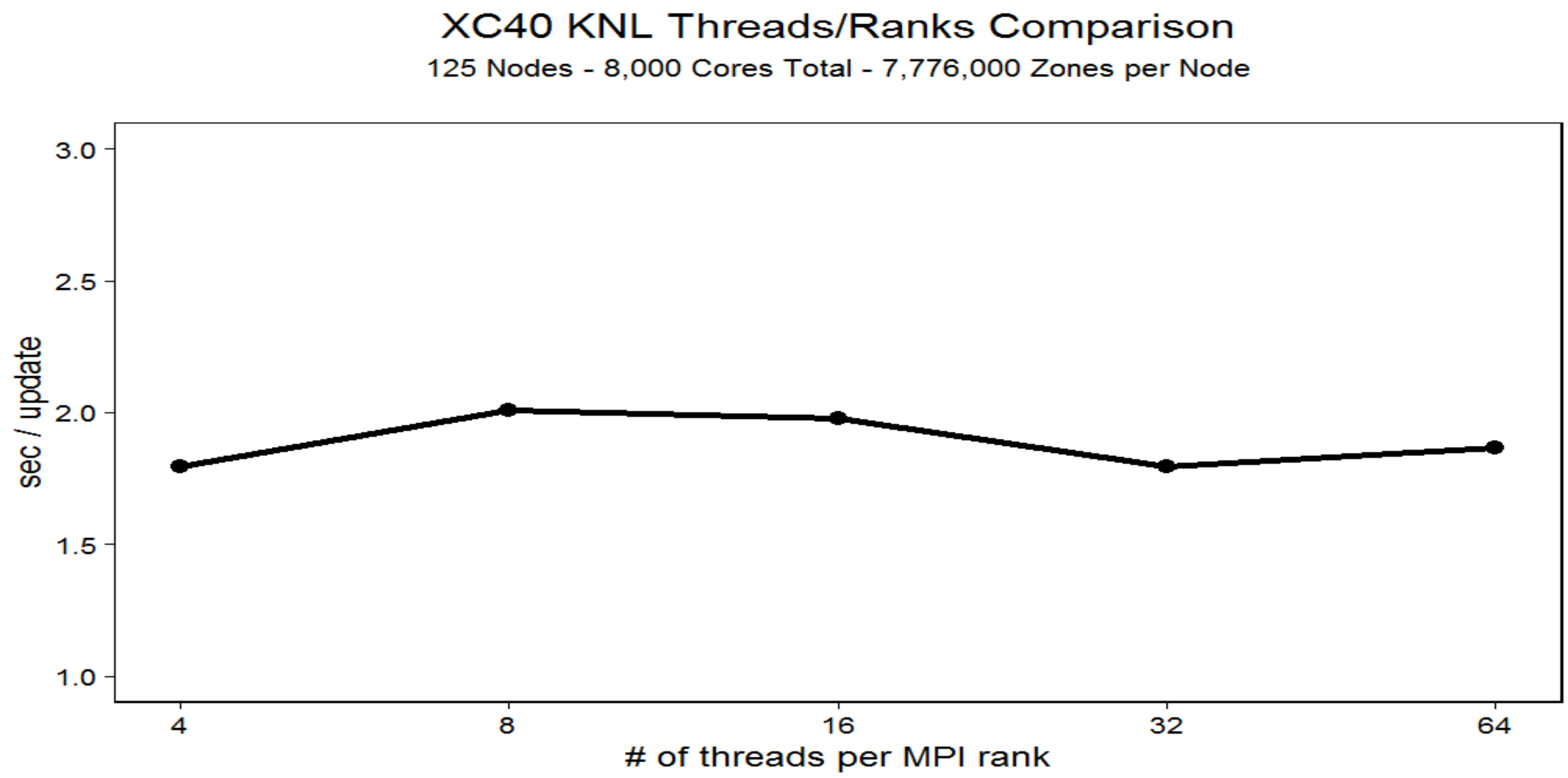


XC40 BDW Weak Scaling

1 rank per node - 36 threads per rank - 7,776,000 zones per rank

- **Rank reordering**
  - Cartesian domain optimization for XC topology/placement improves largest run wall time by additional 2.3%


XC40 KNL Weak Scaling
1 rank per node - 64 threads per rank - 7,776,000 zones per rank
Configuration
global lock MPI-RMA (MPT 7.3.1)
thread hot MPI-RMA (MPT 7.3.2)
sec / update
2.25
2.00
1.75
128
512
2048
8192
# of cores



- Less than 5% difference between 4 and 64 threads per rank
- Ideal for application like Wombat is 0%

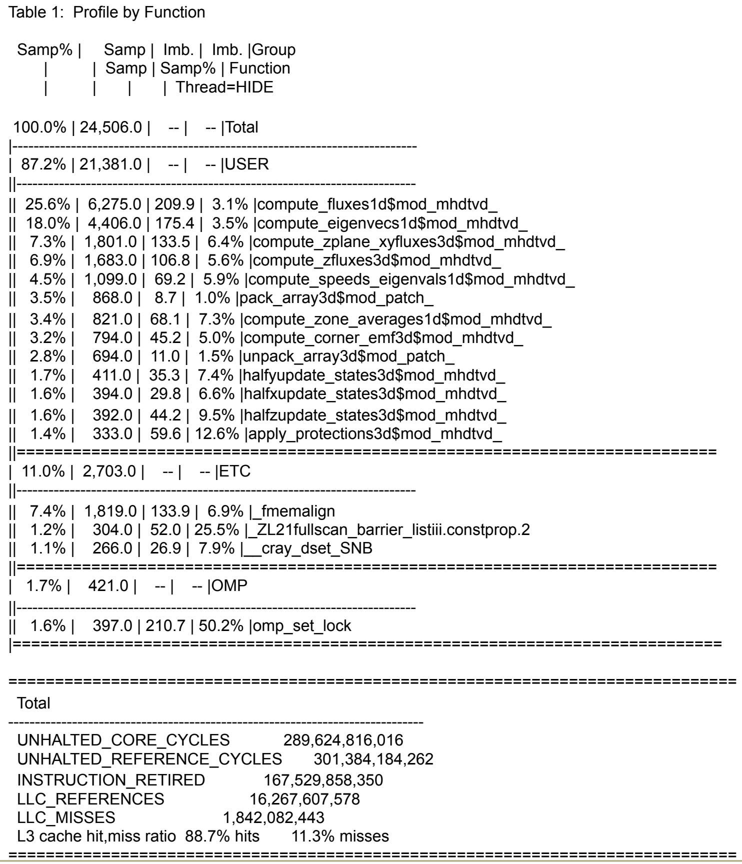
Single KNL Node Profile

- 68 threads
- 17x8x4 Patch domain (30x30x30 zones per Patches)
- HiMem = 16,076 MB
- 2.97 s/update

```
Table 1:  Profile by Function

  Samp% |    Samp |  Imb. |  Imb. |Group
        |         |  Samp | Samp% | Function
        |         |       |       |  Thread=HIDE

 100.0% | 3,487.0 |    -- |    -- |Total
|------------------------------------------------------------------------------
|  75.7% | 2,639.0 |    -- |    -- |USER
||-----------------------------------------------------------------------------
||  22.8% |   795.0 |  61.7 |  7.3% |compute_fluxes1d$mod_mhdtvd_
||  16.4% |   572.0 |  50.2 |  7.5% |compute_eigenvecs1d$mod_mhdtvd_
||   7.0% |   244.0 |  37.7 | 13.2% |compute_zplane_xyfluxes3d$mod_mhdtvd_
||   6.6% |   231.0 |  22.9 |  9.0% |compute_zfluxes3d$mod_mhdtvd_
||   4.0% |   140.0 |  20.4 | 13.3% |compute_speeds_eigenvals1d$mod_mhdtvd_
||   3.0% |   103.0 |  15.4 | 13.2% |compute_corner_emf3d$mod_mhdtvd_
||   2.9% |   101.0 |  23.0 | 18.2% |compute_zone_averages1d$mod_mhdtvd_
||   1.8% |    63.0 |   3.9 |  5.7% |pack_array3d$mod_patch_
||   1.6% |    56.0 |   8.4 | 12.8% |halfyupdate_states3d$mod_mhdtvd_
||   1.5% |    53.0 |  12.2 | 18.8% |halfxupdate_states3d$mod_mhdtvd_
||   1.3% |    45.0 |   8.6 | 13.5% |halfzupdate_states3d$mod_mhdtvd_
||   1.0% |    35.0 |  13.7 | 25.2% |apply_protections3d$mod_mhdtvd_
||=============================================================================
|  19.3% |   672.0 |    -- |    -- |ETC
||-----------------------------------------------------------------------------
||   7.1% |   248.0 |   1.0 |  0.8% |_ZL21fullscan_barrier_listiii.constprop.2
||   6.0% |   210.0 |  27.0 | 11.6% |_fmemalign
||   1.3% |    46.0 |  33.0 | 58.9% |_ZL23internal_simple_barrierii.constprop.3
||   1.2% |    42.0 |  15.9 | 28.3% |__cray_dset_SNB
||   1.1% |    39.0 |    -- |    -- |_dl_update_slotinfo
||=============================================================================
|   5.0% |   174.0 |    -- |    -- |OMP
||-----------------------------------------------------------------------------
||   4.3% |   151.0 | 194.2 | 49.7% |omp_set_lock
|==============================================================================

===============================================================================
  Total
-------------------------------------------------------------------------------
  UNHALTED_CORE_CYCLES                43,959,396,781
  UNHALTED_REFERENCE_CYCLES           43,669,216,374
  INSTRUCTION_RETIRED                 30,208,753,315
  LLC_REFERENCES                       2,616,929,294
  LLC_MISSES                             247,223,361
  L3 cache hit,miss ratio  90.6% hits      9.4% misses
===============================================================================
```

Single KNL Node Profile (8X bigger grid)

- 68 threads
- 17x16x16 Patch domain (30x30x30 zones per Patches)
- HiMem = 71,550 MB
- 23.93 s/update
  - 8.08X longer time steps
- Spilling outside of MCDRAM cache not a performance issue



```
Table 1:  Profile by Function

 Samp% |     Samp |  Imb. |  Imb. |Group
       |          |  Samp | Samp% | Function
       |          |       |       |  Thread=HIDE

100.0% | 24,506.0 |    -- |    -- |Total
|-----------------------------------------------------------------------
| 87.2% | 21,381.0 |    -- |    -- |USER
||----------------------------------------------------------------------
|| 25.6% |  6,275.0 | 209.9 |  3.1% |compute_fluxes1d$mod_mhdtvd_
|| 18.0% |  4,406.0 | 175.4 |  3.5% |compute_eigenvecs1d$mod_mhdtvd_
||  7.3% |  1,801.0 | 133.5 |  6.4% |compute_zplane_xyfluxes3d$mod_mhdtvd_
||  6.9% |  1,683.0 | 106.8 |  5.6% |compute_zfluxes3d$mod_mhdtvd_
||  4.5% |  1,099.0 |  69.2 |  5.9% |compute_speeds_eigenvals1d$mod_mhdtvd_
||  3.5% |    868.0 |   8.7 |  1.0% |pack_array3d$mod_patch_
||  3.4% |    821.0 |  68.1 |  7.3% |compute_zone_averages1d$mod_mhdtvd_
||  3.2% |    794.0 |  45.2 |  5.0% |compute_corner_emf3d$mod_mhdtvd_
||  2.8% |    694.0 |  11.0 |  1.5% |unpack_array3d$mod_patch_
||  1.7% |    411.0 |  35.3 |  7.4% |halfyupdate_states3d$mod_mhdtvd_
||  1.6% |    394.0 |  29.8 |  6.6% |halfxupdate_states3d$mod_mhdtvd_
||  1.6% |    392.0 |  44.2 |  9.5% |halfzupdate_states3d$mod_mhdtvd_
||  1.4% |    333.0 |  59.6 | 12.6% |apply_protections3d$mod_mhdtvd_
||======================================================================
| 11.0% |  2,703.0 |    -- |    -- |ETC
||----------------------------------------------------------------------
||  7.4% |  1,819.0 | 133.9 |  6.9% |_fmemalign
||  1.2% |    304.0 |  52.0 | 25.5% |_ZL21fullscan_barrier_listiii.constprop.2
||  1.1% |    266.0 |  26.9 |  7.9% |__cray_dset_SNB
||======================================================================
|  1.7% |    421.0 |    -- |    -- |OMP
||----------------------------------------------------------------------
||  1.6% |    397.0 | 210.7 | 50.2% |omp_set_lock
|=======================================================================

========================================================================
 Total
------------------------------------------------------------------------
 UNHALTED_CORE_CYCLES             289,624,816,016
 UNHALTED_REFERENCE_CYCLES        301,384,184,262
 INSTRUCTION_RETIRED              167,529,858,350
 LLC_REFERENCES                    16,267,607,578
 LLC_MISSES                         1,842,082,443
 L3 cache hit,miss ratio  88.7% hits      11.3% misses
========================================================================
```

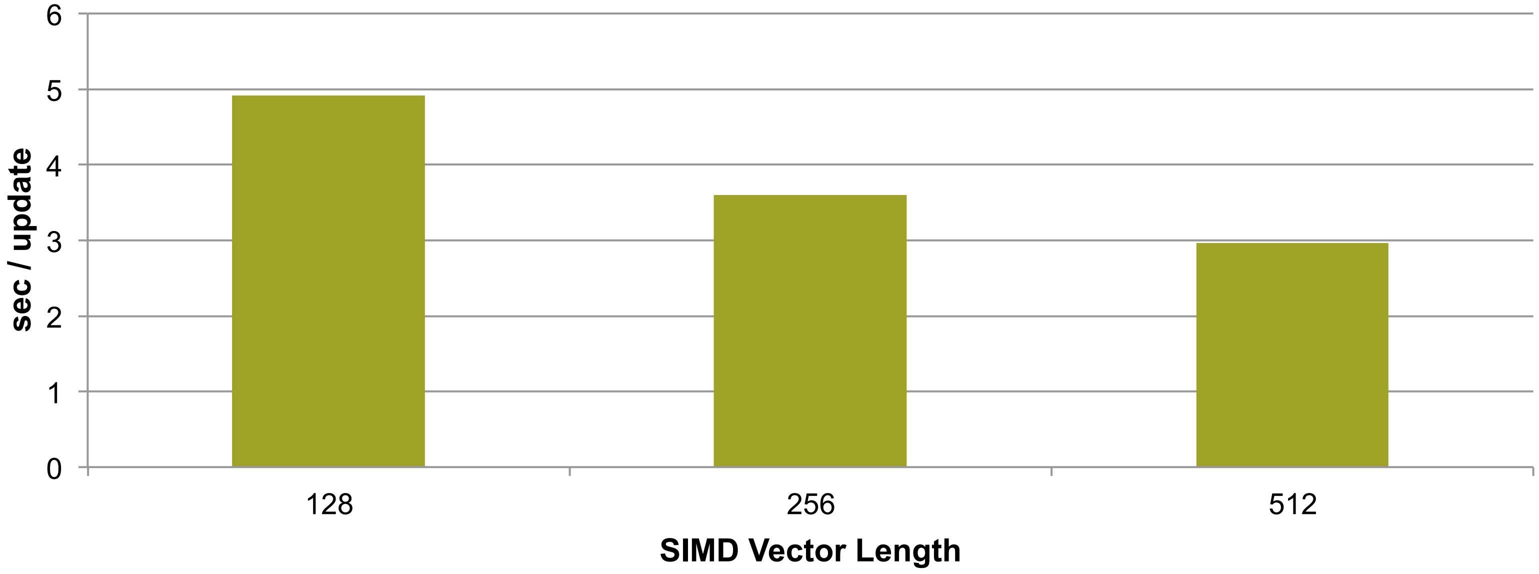
27 KNL Nodes Profile

- 68 threads per rank
- 27 ranks
- 17x16x16 Patch domain (30x30x30 zones per Patches)
- HiMem = 32,678 MB
- 3.43 s/update

Table 1: Profile by Function

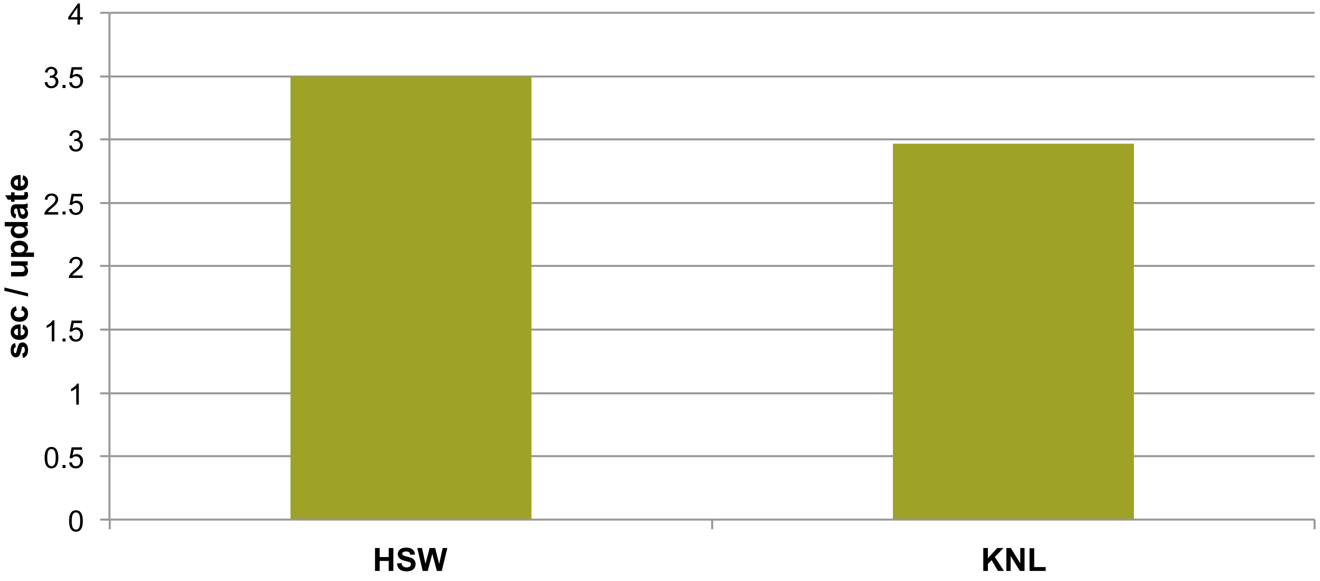
```
 Samp% |    Samp |  Imb. |  Imb. |Group
       |         |  Samp | Samp% | Function
       |         |       |       |  PE=HIDE
       |         |       |       |   Thread=HIDE

100.0% | 4,330.7 |    -- |    -- |Total
|-----------------------------------------------------------------------
|  60.6% | 2,622.6 |    -- |    -- |USER
||----------------------------------------------------------------------
||  16.4% |   710.0 |  49.0 |  6.7% |compute_fluxes1d$mod_mhdtvd_
||  12.0% |   519.8 |  42.2 |  7.8% |compute_eigenvecs1d$mod_mhdtvd_
||   5.2% |   225.8 |   7.2 |  3.2% |initrma$mod_decomposition_
||   5.1% |   220.3 |  32.7 | 13.4% |compute_zplane_xyfluxes3d$mod_mhdtvd_
||   4.7% |   205.3 |  21.7 |  9.9% |compute_zfluxes3d$mod_mhdtvd_
||   2.7% |   117.0 |  19.0 | 14.5% |compute_speeds_eigenvals1d$mod_mhdtvd_
||   2.1% |    91.6 |  26.4 | 23.3% |compute_zone_averages1d$mod_mhdtvd_
||   2.0% |    88.5 |  11.5 | 11.9% |compute_corner_emf3d$mod_mhdtvd_
||   1.2% |    53.9 |   8.1 | 13.6% |pack_array3d$mod_patch_
||   1.1% |    49.7 |   7.3 | 13.2% |halfyupdate_states3d$mod_mhdtvd_
||   1.1% |    45.9 |   7.1 | 13.9% |halfzupdate_states3d$mod_mhdtvd_
||   1.0% |    44.9 |   5.1 | 10.7% |halfxupdate_states3d$mod_mhdtvd_
||======================================================================
|  25.5% | 1,104.1 |    -- |    -- |ETC
||----------------------------------------------------------------------
||  12.8% |   553.3 | 154.7 | 22.7% |_ZL21fullscan_barrier_listiii.constprop.2
||   4.6% |   199.7 |  43.3 | 18.5% |_fmemalign
||   2.8% |   120.8 |  32.2 | 21.8% |_dl_update_slotinfo
||   1.2% |    53.4 |  32.6 | 39.4% |update_get_addr
||======================================================================
|   9.3% |   402.6 |    -- |    -- |OMP
||----------------------------------------------------------------------
||   4.5% |   194.9 | 224.1 | 55.5% |omp_set_lock
||   4.4% |   189.6 |  63.4 | 26.0% |_cray$mt_barrier_part__prime_wait_others
||======================================================================
|   4.6% |   201.3 |    -- |    -- |MPI
||----------------------------------------------------------------------
||   3.9% |   168.0 |  34.0 | 17.5% |mpi_put
|=======================================================================
```

KNL Performance - SIMD Vector Length
14,688,000 zones - 68 threads

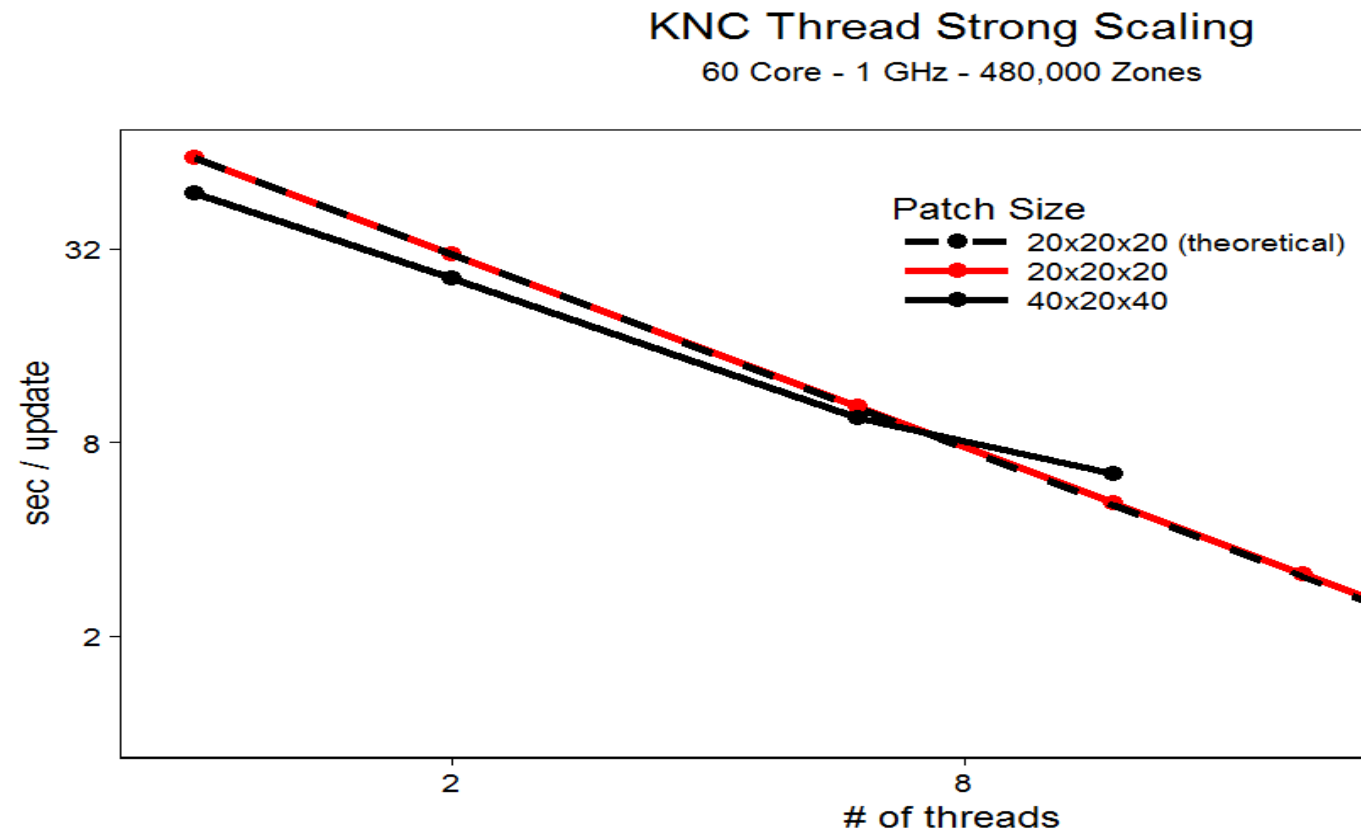


- Vector length is important
- Significant effort went into making solver loops (compute and copy) vectorize

HSW/KNL Node Performance Comparison
14,688,000 zones



- For this problem size a single KNL node is ~15% faster than a HSW node
- Have measured as high as 20% for other problem sizes
- Multi-node comparisons trend similar (KNL is 5-20% faster depending on exact proble


KNC Thread Strong Scaling
60 Core - 1 GHz - 480,000 Zones
Patch Size
20x20x20 (theoretical)
20x20x20
40x20x40
sec / update
32
8
2
# of threads
2
8
32

SPMD OpenMP Pros

- Much less overhead than low level !$OMP PARALLEL DO and !$OMP DO
- Scoping is handled with language standard
- In very complicated applications, the number of directives and code modifications is much less

SPMD OpenMP Cons

- Users must really understand threading and the implications of how the language handles global and local storage which would become shared by all threads and private to a thread.

Analysis of BGW Kernels

A study of higher level OpenMP

Introduction

- BGW test has three kernels
- Reveal does very well on one
- Optimization improved other two significantly

Profiling the BGW test

```
Table 2:  Inclusive and Exclusive Time in Loops (from -hprofile_generate)

  Loop | Loop Incl |      Time |  Loop Hit |    Loop |  Loop |   Loop |Function=/.LOOP[.]
  Incl |      Time |     (Loop |           |   Trips | Trips |  Trips |
 Time% |           |     Adj.) |           |     Avg |   Min |    Max |
|-----------------------------------------------------------------------------
| 94.5% | 55.048452 |  0.003011 |         1 |    96.0 |    96 |     96 |hackakernel_.LOOP.04.li.121
| 91.5% | 53.298164 |  0.015635 |        96 |   240.0 |   240 |    240 |hackakernel_.LOOP.08.li.166
| 91.5% | 53.282529 |  0.055862 |    23,040 |   100.0 |   100 |    100 |hackakernel_.LOOP.09.li.170
| 91.4% | 53.226667 | 53.226667 | 2,304,000 | 8,000.0 | 8,000 |  8,000 |hackakernel_.LOOP.10.li.177
|  2.8% |  1.658162 |  0.713467 |        96 |   240.0 |   240 |    240 |hackakernel_.LOOP.11.li.192
|  1.6% |  0.944695 |  0.944695 |    22,944 | 6,000.0 | 6,000 |  6,000 |hackakernel_.LOOP.12.li.215
|  0.1% |  0.081882 |  0.000093 |        96 |   100.0 |   100 |    100 |hackakernel_.LOOP.06.li.138
|  0.1% |  0.081789 |  0.081789 |     9,600 | 8,000.0 | 8,000 |  8,000 |hackakernel_.LOOP.07.li.145
|  0.0% |  0.005393 |  0.005393 |        96 | 6,000.0 | 6,000 |  6,000 |hackakernel_.LOOP.13.li.243
|  0.0% |  0.000204 |  0.000204 |        96 | 6,000.0 | 6,000 |  6,000 |hackakernel_.LOOP.05.li.129
|  0.0% |  0.000000 |  0.000000 |         1 |   240.0 |   240 |    240 |hackakernel_.LOOP.02.li.79
|  0.0% |  0.000000 |  0.000000 |         1 |   240.0 |   240 |    240 |hackakernel_.LOOP.03.li.85
|  0.0% |  0.000000 |  0.000000 |         1 |    96.0 |    96 |     96 |hackakernel_.LOOP.01.li.60
|=============================================================================
```

Loop B

Loop C

Loop A

Kernel A loop

```
138.   + 1 2---------<          do my_igp = 1, ngpown       (100)
139.     1 2                      if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle
140.     1 2
141.     1 2                      igmax=ncouls
142.     1 2
143.     1 2                      mygpvar1 = CONJG(leftvector(my_igp,n1))
144.     1 2
145.     1 2 Vr2-----<            do ig = 1, igmax             (800)
146.     1 2 Vr2                    matngmatmgpD(ig,my_igp) = rightvector(ig,n1) * mygpvar1
147.     1 2 Vr2----->            enddo
148.     1 2--------->          enddo
```

Which of these two loop should we parallelize?
Are there some inefficiencies?

Kernel B loop

```
165.      1 A-------<>          schdt_array = 0D0
166.    + 1 b--------<          do ifreq=1,nFreq                  (240)
167.      1 b
168.      1 b                         schDt = (0D0,0D0)
169.      1 b
170.    + 1 b b------<                do my_igp = 1, ngpown            (100)
171.      1 b b
172.      1 b b                          if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle
173.      1 b b
174.      1 b b                          igmax=ncouls
175.      1 b b
176.      1 b b                          schDtt = (0D0,0D0)
177.      1 b b Vr3--<                   do ig = 1, igmax                (8000)
178.      1 b b Vr3                         I_epsRggp_int = I_epsR_array(ig,my_igp,ifreq)
179.      1 b b Vr3                         I_epsAggp_int = I_epsA_array(ig,my_igp,ifreq)
180.      1 b b Vr3                         schD=I_epsRggp_int-I_epsAggp_int
181.      1 b b Vr3                         schDtt = schDtt + matngmatmgpD(ig,my_igp)*schD
182.      1 b b Vr3-->                   enddo
183.      1 b b                          schdt_array(ifreq) = schdt_array(ifreq) + schDtt
184.      1 b b------>                enddo
185.      1 b
186.      1 b-------->          enddo
187.      1
188.    + 1                     call timget(endtime_ch)
189.      1                     time_b = time_b + endtime_ch - starttime_ch
190.    + 1                     call timget(starttime_ch)
```

Once again which loop is best to target for parallelization

Kernel C loop

```
192.  + 1 2---------<          do ifreq=1,nFreq                    (240)
193.    1 2
194.    1 2                        schDt = schDt_array(ifreq)
195.    1 2
196.    1 2                        cedifft_zb = dFreqGrid(ifreq)
197.    1 2                        cedifft_coh = CMPLX(cedifft_zb,0D0)- dFreqBrd(ifreq)
198.    1 2
199.    1 2                        if (ifreq .ne. 1) then
200.    1 2                          cedifft_zb_right = cedifft_zb
201.    1 2                          cedifft_zb_left = dFreqGrid(ifreq-1)
202.    1 2                          schDt_right = schDt
203.    1 2                          schDt_left = schDt_array(ifreq-1)
204.    1 2                          schDt_avg = 0.5D0 * ( schDt_right + schDt_left )
205.    1 2                          schDt_lin = schDt_right - schDt_left
206.    1 2                          schDt_lin2 = schDt_lin/(cedifft_zb_right-cedifft_zb_left)
207.    1 2                        endif
208.    1 2
209.    1 2           ! The below two lines are for sigma1 and sigma3
210.    1 2                        if (ifreq .ne. nFreq) then
211.    1 2 fVr2--<>                 schDi(:) = schDi(:) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(:)-cedifft_coh)
212.    1 2 f-----<>                 schDi_corb(:) = schDi_corb(:) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(:)-cedif
213.    1 2                        endif
214.    1 2                        if (ifreq .ne. 1) then
215.    1 2 V------<                 do iw = 1, nfreqeval            (6000)
```

Once again which loop is best to target for parallelization

Array Assignment

Kernel C loop -continued

```
216.    1 2 V        !These lines are for sigma2
217.    1 2 V                        intfact=abs((wxi(iw)-cedifft_zb_right)/(wxi(iw)-cedifft_zb_left))
218.    1 2 V                        if (intfact .lt. 1d-4) intfact = 1d-4
219.    1 2 V                        if (intfact .gt. 1d4) intfact = 1d4
220.    1 2 V                        intfact = -log(intfact)
221.    1 2 V                        sch2Di(iw) = sch2Di(iw) - CMPLX(0.d0,prefactor) * schDt_avg * intfact
222.    1 2 V        !These lines are for sigma4
223.    1 2 V                        if (flag_occ) then
224.    1 2 V                          intfact=abs((wxi(iw)+cedifft_zb_right)/(wxi(iw)+cedifft_zb_left))
225.    1 2 V                          if (intfact .lt. 1d-4) intfact = 1d-4
226.    1 2 V                          if (intfact .gt. 1d4) intfact = 1d4
227.    1 2 V                          intfact = log(intfact)
228.    1 2 V                          schDt_lin3 = (schDt_left + schDt_lin2*(-wxi(iw)-cedifft_zb_left))*intfact
229.    1 2 V                        else
230.    1 2 V                          schDt_lin3 = (schDt_left + schDt_lin2*(wxi(iw)-cedifft_zb_left))*intfact
231.    1 2 V                        endif
232.    1 2 V                        schDt_lin3 = schDt_lin3 + schDt_lin
233.    1 2 V                        schDi_cor(iw) = schDi_cor(iw) - CMPLX(0.d0,prefactor) * schDt_lin3
234.    1 2 V------>               enddo
235.    1 2                      endif
236.    1 2-------->         enddo
```

Kernel A

```
 do my_igp = 1, ngpown
  if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle

  igmax=ncouls

  mygpvar1 = CONJG(leftvector(my_igp,n1))

  do ig = 1, igmax
    matngmatmgpD(ig,my_igp) = rightvector(ig,n1) * mygpvar1
  enddo
enddo
```

Kernel A

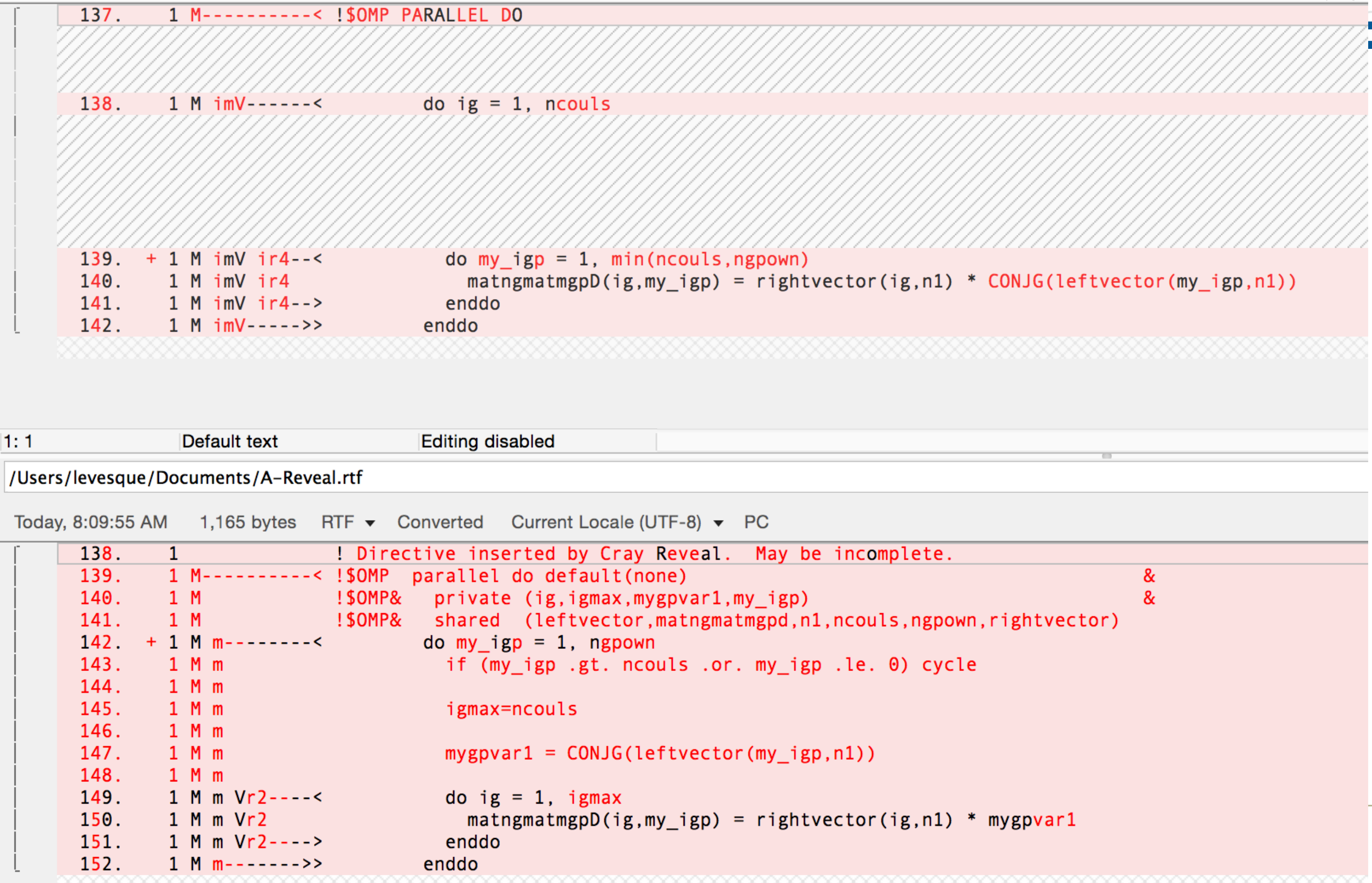
Automatically inserted by Reveal

```
138.      1              ! Directive inserted by Cray Reveal.  May be incomplete.
139.      1 M----------< !$OMP  parallel do default(none)                                  &
140.      1 M            !$OMP&   private (ig,igmax,mygpvar1,my_igp)                       &
141.      1 M            !$OMP&   shared  (leftvector,matngmatmgpd,n1,ncouls,ngpown,rightvector)
142.    + 1 M m--------<          do my_igp = 1, ngpown
143.      1 M m                     if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle
144.      1 M m
145.      1 M m                     igmax=ncouls
146.      1 M m
147.      1 M m                     mygpvar1 = CONJG(leftvector(my_igp,n1))
148.      1 M m
149.      1 M m Vr2----<            do ig = 1, igmax
150.      1 M m Vr2                   matngmatmgpD(ig,my_igp) = rightvector(ig,n1) * mygpvar1
151.      1 M m Vr2---->            enddo
152.      1 M m------->>          enddo
```

Kernel A

Automatically changed by Levesque

```
137.       1 M-----------< !$OMP PARALLEL DO
138.       1 M imV-------<         do ig = 1, ncouls
139.     + 1 M imV ir4--<            do my_igp = 1, min(ncouls,ngpown)
140.       1 M imV ir4                 matngmatmgpD(ig,my_igp) = rightvector(ig,n1) * CONJG(leftvector(my_igp,n1))
141.       1 M imV ir4-->            enddo
142.       1 M imV------>>         enddo
```



```
137.       1 M----------< !$OMP PARALLEL DO

138.       1 M imV------<          do ig = 1, ncouls

139.     + 1 M imV ir4--<            do my_igp = 1, min(ncouls,ngpown)
140.       1 M imV ir4                 matngmatmgpD(ig,my_igp) = rightvector(ig,n1) * CONJG(leftvector(my_igp,n1))
141.       1 M imV ir4-->            enddo
142.       1 M imV----->>          enddo
```

/Users/levesque/Documents/A-Reveal.rtf

```
138.       1               ! Directive inserted by Cray Reveal.  May be incomplete.
139.       1 M----------< !$OMP  parallel do default(none)                                   &
140.       1 M            !$OMP&   private (ig,igmax,mygpvar1,my_igp)                         &
141.       1 M            !$OMP&   shared  (leftvector,matngmatmgpd,n1,ncouls,ngpown,rightvector)
142.     + 1 M m--------<          do my_igp = 1, ngpown
143.       1 M m                     if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle
144.       1 M m
145.       1 M m                     igmax=ncouls
146.       1 M m
147.       1 M m                     mygpvar1 = CONJG(leftvector(my_igp,n1))
148.       1 M m
149.       1 M m Vr2----<            do ig = 1, igmax
150.       1 M m Vr2                   matngmatmgpD(ig,my_igp) = rightvector(ig,n1) * mygpvar1
151.       1 M m Vr2---->            enddo
152.       1 M m------->>          enddo
```

Kernel B

```
schdt_array = 0D0
do ifreq=1,nFreq

    schDt = (0D0,0D0)

    do my_igp = 1, ngpown

      if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle

      igmax=ncouls

      schDtt = (0D0,0D0)
      do ig = 1, igmax
        I_epsRggp_int = I_epsR_array(ig,my_igp,ifreq)
        I_epsAggp_int = I_epsA_array(ig,my_igp,ifreq)
        schD=I_epsRggp_int-I_epsAggp_int
        schDtt = schDtt + matngmatmgpD(ig,my_igp)*schD
      enddo
      schdt_array(ifreq) = schdt_array(ifreq) + schDtt
    enddo

enddo
```

Kernel B

Automatically inserted by Reveal

```
170.    1              ! Directive inserted by Cray Reveal.  May be incomplete.
171.    1 M----------< !$OMP  parallel do default(none)                                         &
172.    1 M            !$OMP&   private (ifreq,ig,igmax,i_epsaggp_int,i_epsrggp_int,my_igp,     &
173.    1 M            !$OMP&            schd,schdt,schdtt)                                     &
174.    1 M            !$OMP&   shared  (i_epsa_array,i_epsr_array,matngmatmgpd,ncouls,nfreq,   &
175.    1 M            !$OMP&            ngpown,schdt_array)
176.  + 1 M m--------<         do ifreq=1,nFreq
177.    1 M m
178.    1 M m                      schDt = (0D0,0D0)
179.    1 M m
180.  + 1 M m 4------<             do my_igp = 1, ngpown
181.    1 M m 4
182.    1 M m 4                      if (my_igp .gt. ncouls .or. my_igp .le. 0) cycle
183.    1 M m 4
184.    1 M m 4                      igmax=ncouls
185.    1 M m 4
186.    1 M m 4                      schDtt = (0D0,0D0)
187.    1 M m 4 Vr3--<               do ig = 1, igmax
188.    1 M m 4 Vr3                    I_epsRggp_int = I_epsR_array(ig,my_igp,ifreq)
189.    1 M m 4 Vr3                    I_epsAggp_int = I_epsA_array(ig,my_igp,ifreq)
190.    1 M m 4 Vr3                    schD=I_epsRggp_int-I_epsAggp_int
191.    1 M m 4 Vr3                    schDtt = schDtt + matngmatmgpD(ig,my_igp)*schD
192.    1 M m 4 Vr3-->               enddo
193.    1 M m 4                      schdt_array(ifreq) = schdt_array(ifreq) + schDtt
194.    1 M m 4------>             enddo
195.    1 M m
196.    1 M m------->>         enddo
```

Kernel C

```
            do ifreq=1,nFreq

            schDt = schDt_array(ifreq)

            cedifft_zb = dFreqGrid(ifreq)
            cedifft_coh = CMPLX(cedifft_zb,0D0)- dFreqBrd(ifreq)

            if (ifreq .ne. 1) then
              cedifft_zb_right = cedifft_zb
              cedifft_zb_left = dFreqGrid(ifreq-1)
              schDt_right = schDt
              schDt_left = schDt_array(ifreq-1)
              schDt_avg = 0.5D0 * ( schDt_right + schDt_left )
              schDt_lin = schDt_right - schDt_left
              schDt_lin2 = schDt_lin/(cedifft_zb_right-cedifft_zb_left)
            endif

! The below two lines are for sigma1 and sigma3
            if (ifreq .ne. nFreq) then
              schDi(:) = schDi(:) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(:)-cedifft_coh)
              schDi_corb(:) = schDi_corb(:) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(:)-cedifft_cor)
            endif
            if (ifreq .ne. 1) then
              do iw = 1, nfreqeval
!These lines are for sigma2
                intfact=abs((wxi(iw)-cedifft_zb_right)/(wxi(iw)-cedifft_zb_left))
                if (intfact .lt. 1d-4) intfact = 1d-4
                if (intfact .gt. 1d4) intfact = 1d4
                intfact = -log(intfact)
                sch2Di(iw) = sch2Di(iw) - CMPLX(0.d0,prefactor) * schDt_avg * intfact
```

Kernel C

```
!These lines are for sigma4
                if (flag_occ) then
                  intfact=abs((wxi(iw)+cedifft_zb_right)/(wxi(iw)+cedifft_zb_left))
                  if (intfact .lt. 1d-4) intfact = 1d-4
                  if (intfact .gt. 1d4) intfact = 1d4
                  intfact = log(intfact)
                  schDt_lin3 = (schDt_left + schDt_lin2*(-wxi(iw)-cedifft_zb_left))*intfact
                else
                  schDt_lin3 = (schDt_left + schDt_lin2*(wxi(iw)-cedifft_zb_left))*intfact
                endif
                schDt_lin3 = schDt_lin3 + schDt_lin
                schDi_cor(iw) = schDi_cor(iw) - CMPLX(0.d0,prefactor) * schDt_lin3
              enddo
            endif
        enddo
```

Kernel C

```
202.  + 1 2----------<         do ifreq=1,nFreq
203.    1 2
204.    1 2                        schDt = schDt_array(ifreq)
205.    1 2
206.    1 2                        cedifft_zb = dFreqGrid(ifreq)
207.    1 2                        cedifft_coh = CMPLX(cedifft_zb,0D0)- dFreqBrd(ifreq)
208.    1 2
209.    1 2                        if (ifreq .ne. 1) then
210.    1 2                          cedifft_zb_right = cedifft_zb
211.    1 2                          cedifft_zb_left = dFreqGrid(ifreq-1)
212.    1 2                          schDt_right = schDt
213.    1 2                          schDt_left = schDt_array(ifreq-1)
214.    1 2                          schDt_avg = 0.5D0 * ( schDt_right + schDt_left )
215.    1 2                          schDt_lin = schDt_right - schDt_left
216.    1 2                          schDt_lin2 = schDt_lin/(cedifft_zb_right-cedifft_zb_left)
217.    1 2                        endif
218.    1 2
219.    1 2            ! The below two lines are for sigma1 and sigma3
220.    1 2                        if (ifreq .ne. nFreq) then
221.    1 2 fVr2----<>               schDi(:) = schDi(:) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(:)-cedifft_coh)
222.    1 2 f-------<>               schDi_corb(:) = schDi_corb(:) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(:)-cedifft_cor)
223.    1 2                        endif
224.    1 2                        if (ifreq .ne. 1) then
```

Kernel C

Automatically inserted by Reveal

```
225.     1 2             ! Directive inserted by Cray Reveal.  May be incomplete.
226.     1 2 M--------< !$OMP  parallel do default(none)                                &
227.     1 2 M          !$OMP&   private (intfact,iw,schdt_lin3)                        &
228.     1 2 M          !$OMP&   shared  (cedifft_zb_left,cedifft_zb_right,flag_occ,nfreqeval,   &
229.     1 2 M          !$OMP&            prefactor,sch2di,schdi_cor,schdt_avg,schdt_left,      &
230.     1 2 M          !$OMP&            schdt_lin,schdt_lin2,wxi)
231.     1 2 M mV-----<                do iw = 1, nfreqeval
232.     1 2 M mV       !These lines are for sigma2
233.     1 2 M mV                        intfact=abs((wxi(iw)-cedifft_zb_right)/(wxi(iw)-cedifft_zb_left))
234.     1 2 M mV                        if (intfact .lt. 1d-4) intfact = 1d-4
235.     1 2 M mV                        if (intfact .gt. 1d4) intfact = 1d4
236.     1 2 M mV                        intfact = -log(intfact)
237.     1 2 M mV                        sch2Di(iw) = sch2Di(iw) - CMPLX(0.d0,prefactor) * schDt_avg * intfact
238.     1 2 M mV       !These lines are for sigma4
239.     1 2 M mV                        if (flag_occ) then
240.     1 2 M mV                          intfact=abs((wxi(iw)+cedifft_zb_right)/(wxi(iw)+cedifft_zb_left))
241.     1 2 M mV                          if (intfact .lt. 1d-4) intfact = 1d-4
242.     1 2 M mV                          if (intfact .gt. 1d4) intfact = 1d4
243.     1 2 M mV                          intfact = log(intfact)
244.     1 2 M mV                          schDt_lin3 = (schDt_left + schDt_lin2*(-wxi(iw)-cedifft_zb_left))*intfact
245.     1 2 M mV                        else
246.     1 2 M mV                          schDt_lin3 = (schDt_left + schDt_lin2*(wxi(iw)-cedifft_zb_left))*intfact
247.     1 2 M mV                        endif
248.     1 2 M mV                        schDt_lin3 = schDt_lin3 + schDt_lin
249.     1 2 M mV                        schDi_cor(iw) = schDi_cor(iw) - CMPLX(0.d0,prefactor) * schDt_lin3
250.     1 2 M mV---->>                enddo
251.     1 2                         endif
252.     1 2---------->          enddo
```

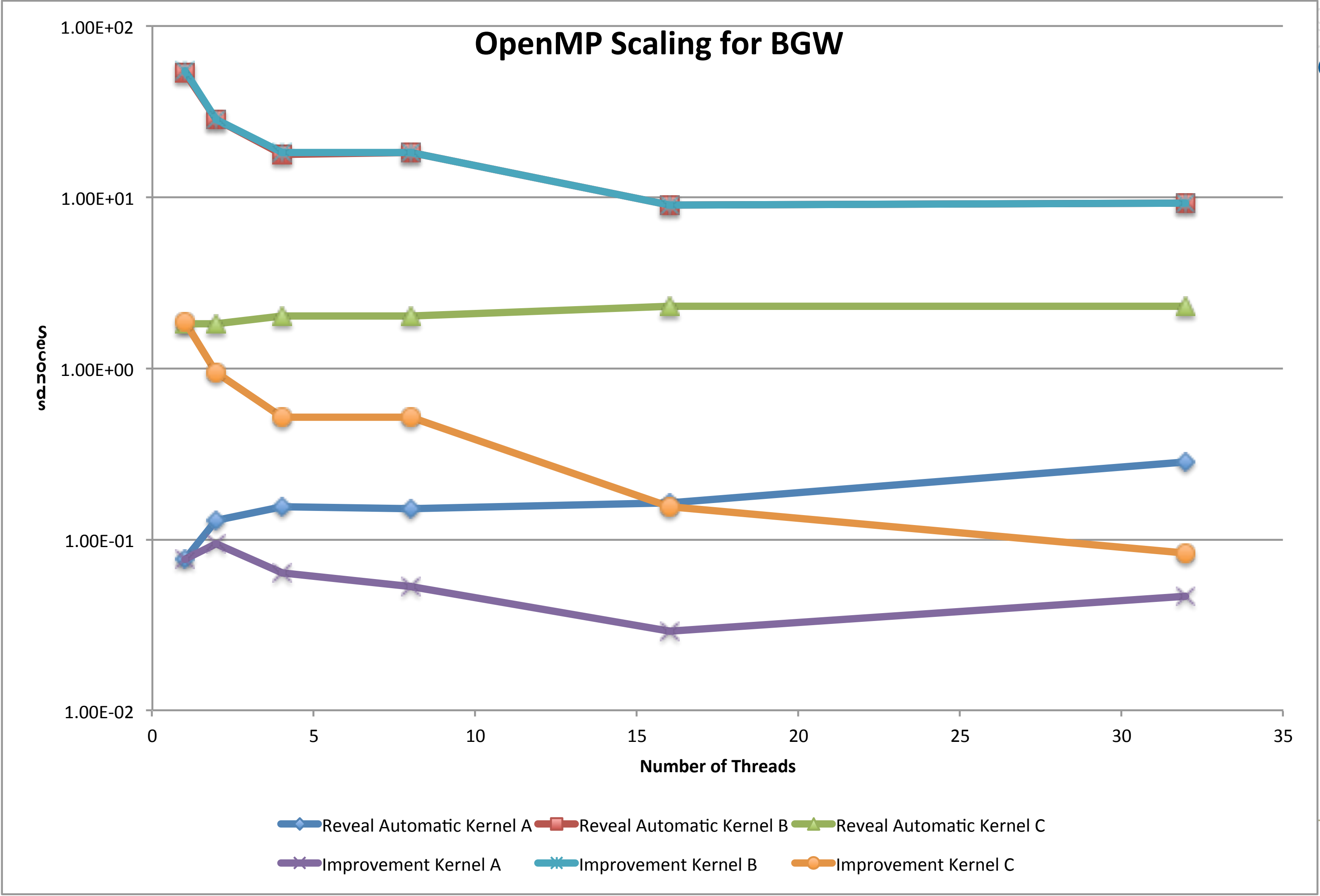
Kernel C

Automatically modified by Levesque

```
1              ! Directive inserted by Cray Reveal.  May be incomplete.
1 M----------< !$OMP  parallel do default(none)                                          &
1 M            !$OMP&   private (cedifft_coh,cedifft_zb,cedifft_zb_left,                 &
1 M            !$OMP&            cedifft_zb_right,ifreq,intfact,iw,schdt,schdt_avg,      &
1 M            !$OMP&            schdt_left,schdt_lin,schdt_lin2,schdt_lin3,             &
1 M            !$OMP&            schdt_right)                                            &
1 M            !$OMP&   shared  (cedifft_cor,dfreqbrd,dfreqgrid,flag_occ,nfreq,          &
1 M            !$OMP&            nfreqeval,pref,prefactor,sch2di,schdi,schdi_cor,        &
1 M            !$OMP&            schdi_corb,schdt_array,wxi)
1 M mV-------<         do iw = 1, nfreqeval
1 M mV 4-----<           do ifreq=1,nFreq
1 M mV 4
1 M mV 4                   schDt = schDt_array(ifreq)
1 M mV 4
1 M mV 4                   cedifft_zb = dFreqGrid(ifreq)
1 M mV 4                   cedifft_coh = CMPLX(cedifft_zb,0D0)- dFreqBrd(ifreq)
1 M mV 4
1 M mV 4                   if (ifreq .ne. 1) then
1 M mV 4                     cedifft_zb_right = cedifft_zb
1 M mV 4                     cedifft_zb_left = dFreqGrid(ifreq-1)
1 M mV 4                     schDt_right = schDt
1 M mV 4                     schDt_left = schDt_array(ifreq-1)
1 M mV 4                     schDt_avg = 0.5D0 * ( schDt_right + schDt_left )
1 M mV 4                     schDt_lin = schDt_right - schDt_left
1 M mV 4                     schDt_lin2 = schDt_lin/(cedifft_zb_right-cedifft_zb_left)
1 M mV 4                   endif
1 M mV 4
1 M mV 4       ! The below two lines are for sigma1 and sigma3
1 M mV 4                   if (ifreq .ne. nFreq) then
1 M mV 4                     schDi(iw) = schDi(iw) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(iw)-cedifft_coh)
1 M mV 4                     schDi_corb(iw) = schDi_corb(iw) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(iw)-cedifft_cor)
1 M mV 4                   endif
1 M mV 4                   if(ifreq.ne.1)then
```

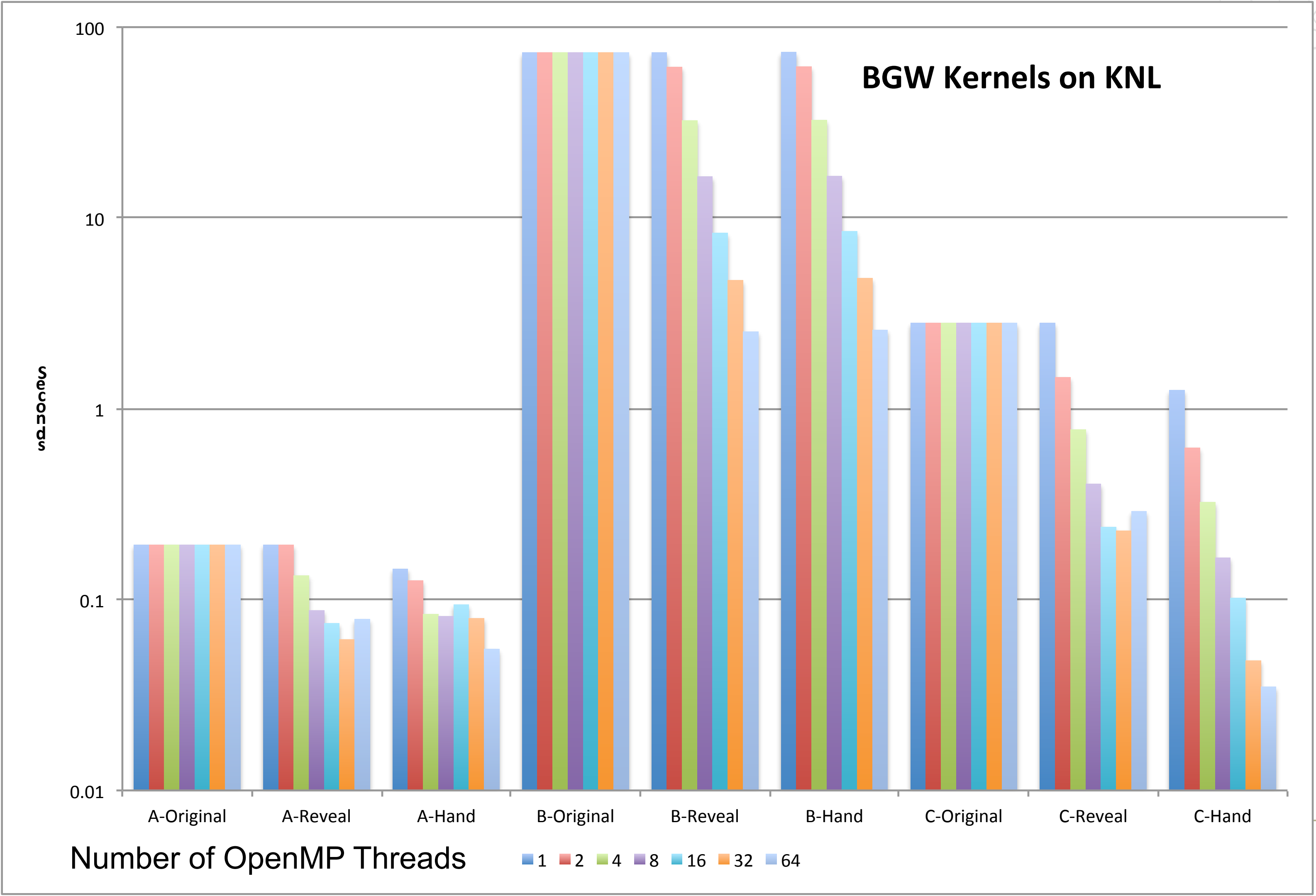
Kernel C

```
219.    1 M mV 4       ! The below two lines are for sigma1 and sigma3
220.    1 M mV 4                   if (ifreq .ne. nFreq) then
221.    1 M mV 4                     schDi(iw) = schDi(iw) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(iw)-cedifft_coh)
222.    1 M mV 4                     schDi_corb(iw) = schDi_corb(iw) - CMPLX(0.d0,pref(ifreq)) * schDt / ( wxi(iw)-cedifft_cor)
223.    1 M mV 4                   endif
224.    1 M mV 4                   if(ifreq.ne.1)then
225.    1 M mV 4       !These lines are for sigma2
226.    1 M mV 4                       intfact=abs((wxi(iw)-cedifft_zb_right)/(wxi(iw)-cedifft_zb_left))
227.    1 M mV 4                       if (intfact .lt. 1d-4) intfact = 1d-4
228.    1 M mV 4                       if (intfact .gt. 1d4) intfact = 1d4
229.    1 M mV 4                       intfact = -log(intfact)
230.    1 M mV 4                       sch2Di(iw) = sch2Di(iw) - CMPLX(0.d0,prefactor) * schDt_avg * intfact
231.    1 M mV 4       !These lines are for sigma4
232.    1 M mV 4                       if (flag_occ) then
233.    1 M mV 4                         intfact=abs((wxi(iw)+cedifft_zb_right)/(wxi(iw)+cedifft_zb_left))
234.    1 M mV 4                         if (intfact .lt. 1d-4) intfact = 1d-4
235.    1 M mV 4                         if (intfact .gt. 1d4) intfact = 1d4
236.    1 M mV 4                         intfact = log(intfact)
237.    1 M mV 4                         schDt_lin3 = (schDt_left + schDt_lin2*(-wxi(iw)-cedifft_zb_left))*intfact
238.    1 M mV 4                       else
239.    1 M mV 4                         schDt_lin3 = (schDt_left + schDt_lin2*(wxi(iw)-cedifft_zb_left))*intfact
240.    1 M mV 4                       endif
241.    1 M mV 4                       schDt_lin3 = schDt_lin3 + schDt_lin
242.    1 M mV 4                       schDi_cor(iw) = schDi_cor(iw) - CMPLX(0.d0,prefactor) * schDt_lin3
243.    1 M mV 4                   endif
244.    1 M mV 4----->           enddo
245.    1 M mV------>>         enddo
```

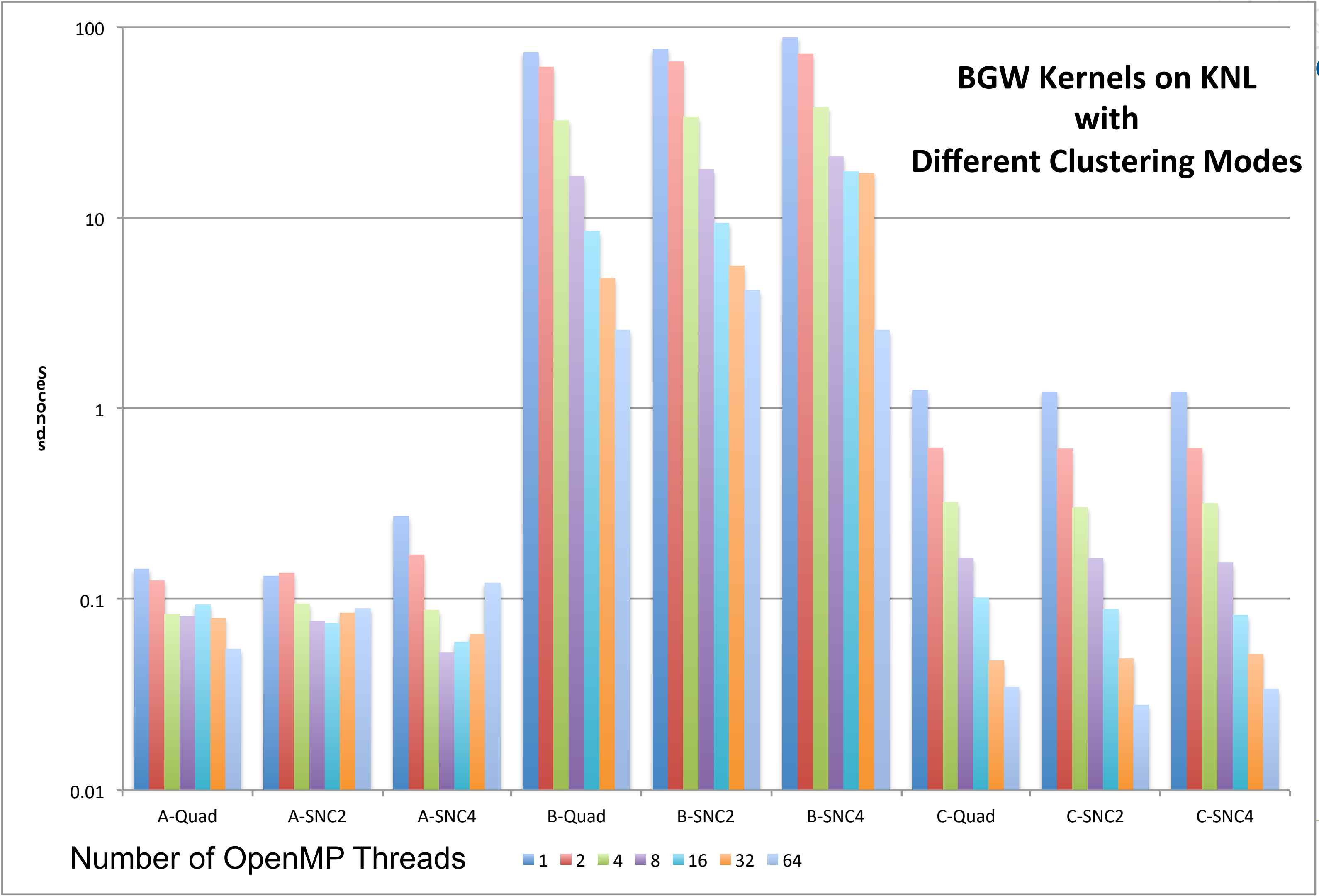


OpenMP Scaling for BGW

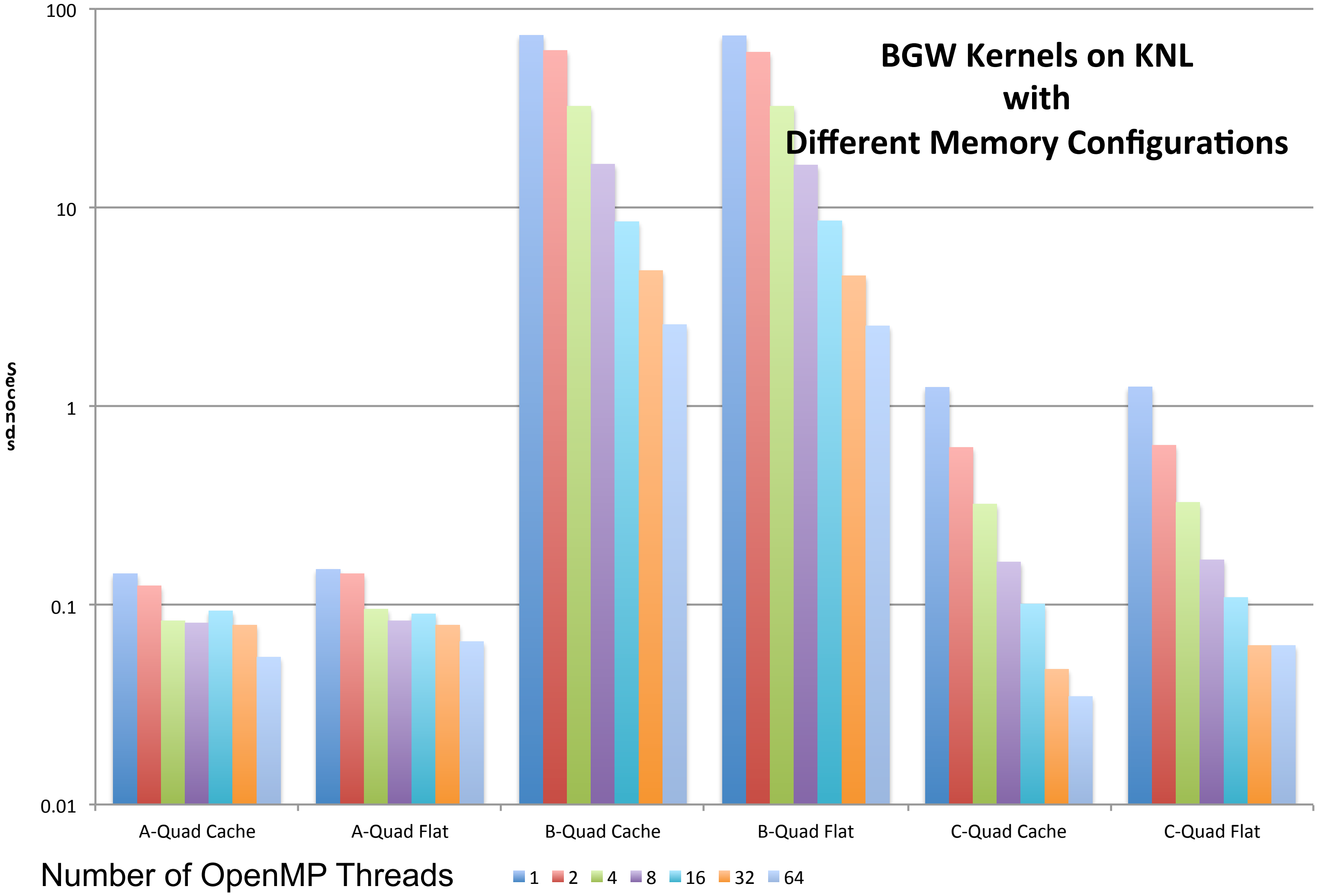
BGW Kernels on KNL



BGW Kernels on KNL with Different Clustering Modes

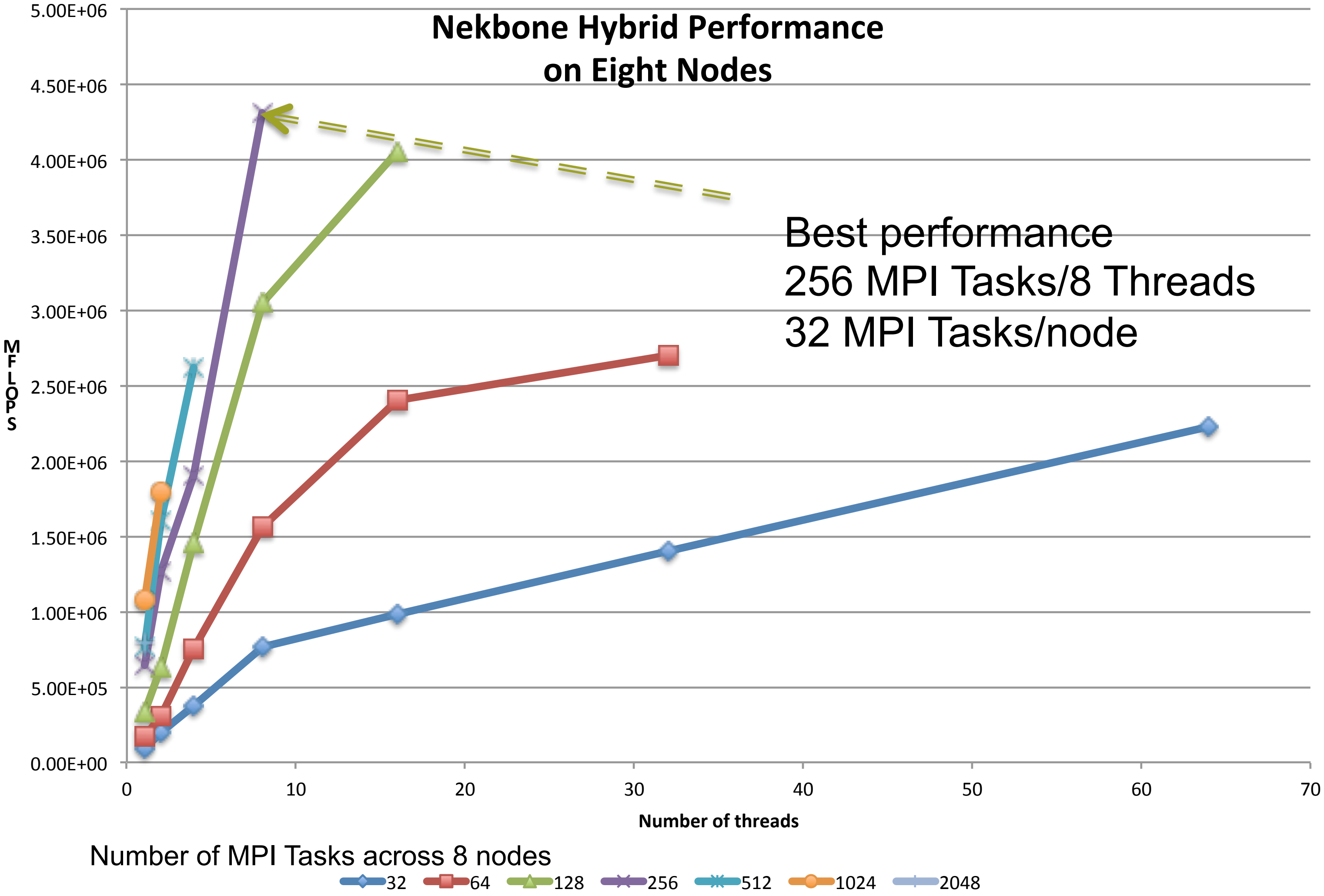


BGW Kernels on KNL with Different Memory Configurations



Nekbone

High order, incompressible Navier-Stokes solver based on the spectral element method



Nekbone Hybrid Performance on Eight Nodes

Best performance
256 MPI Tasks/8 Threads
32 MPI Tasks/node

Comment on Application

- **Excellent OpenMP**
  - SPMD style with consistent memory access pattern
    - Come to CUG and see my tutorial
  - Only example that shows Hybrid a clear winner over all MPI

S3D

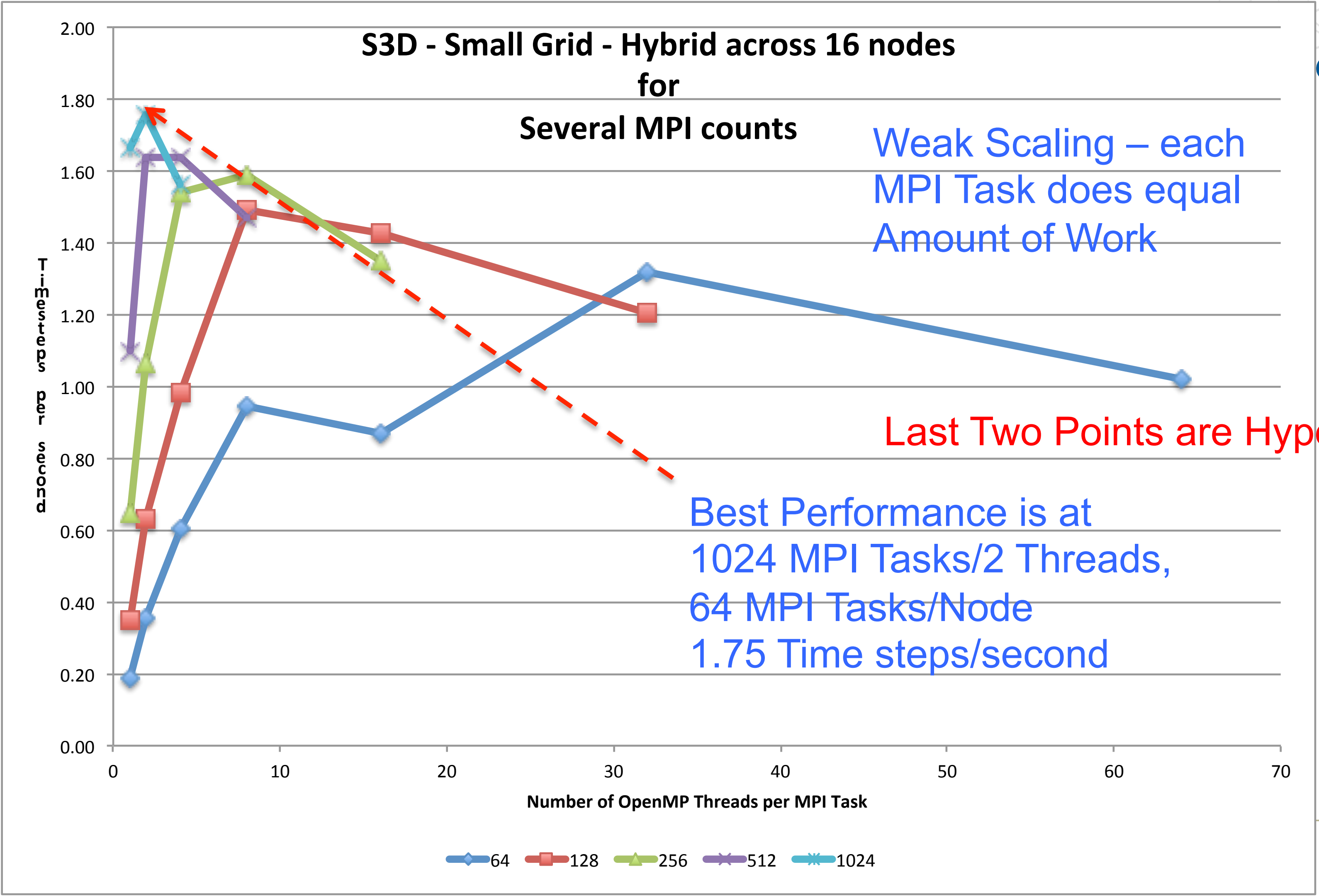
Flow solver for performing direct numerical simulation (DNS) of turbulent combustion.

Details of Test

- **OpenACC/OpenMP code written for Titan**
  - Zero modifications from Titan code to KNL
  - OpenACC – OpenMP controlled by #IFDEF
- **Test Problems (Small and Medium fit in MCDRAM)**
  - Small Grid (6.5 GB/node)
    - Size of Titan Acceptance test
    - 192*192*192 on 16 nodes
  - Medium Grid (10 GB/node)
    - 384*384*192 on 16 nodes
  - Large Grid (24 GB/node)
    - 768*384*384 on 16 nodes
- **Ran on 8,16,32,64 ,128 and 256 nodes**
  - Weak Scaling employed for number of MPI tasks

Comment on Application

- **Reasonably Good OpenMP**
  - It is the traditional approach with high level Parallel DO
  - One Parallel loop may not have same memory access pattern as the next
    - Could cause caching issues
  - Could be improved with SPMD style OpenMP to maintain constant memory access across parallel loops

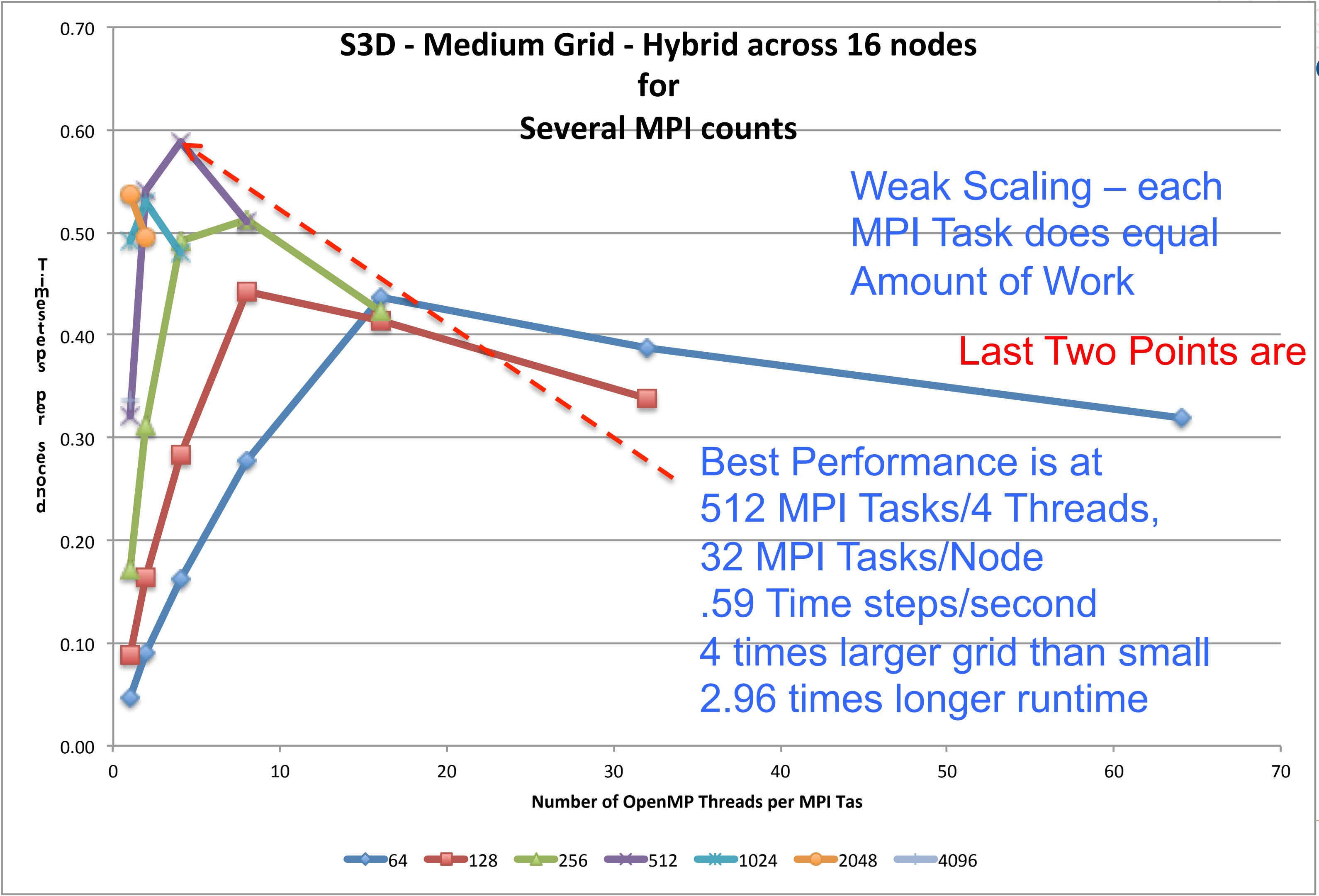


S3D - Small Grid - Hybrid across 16 nodes for Several MPI counts

Weak Scaling – each MPI Task does equal Amount of Work

Last Two Points are Hyper-threads

Best Performance is at
1024 MPI Tasks/2 Threads,
64 MPI Tasks/Node
1.75 Time steps/second

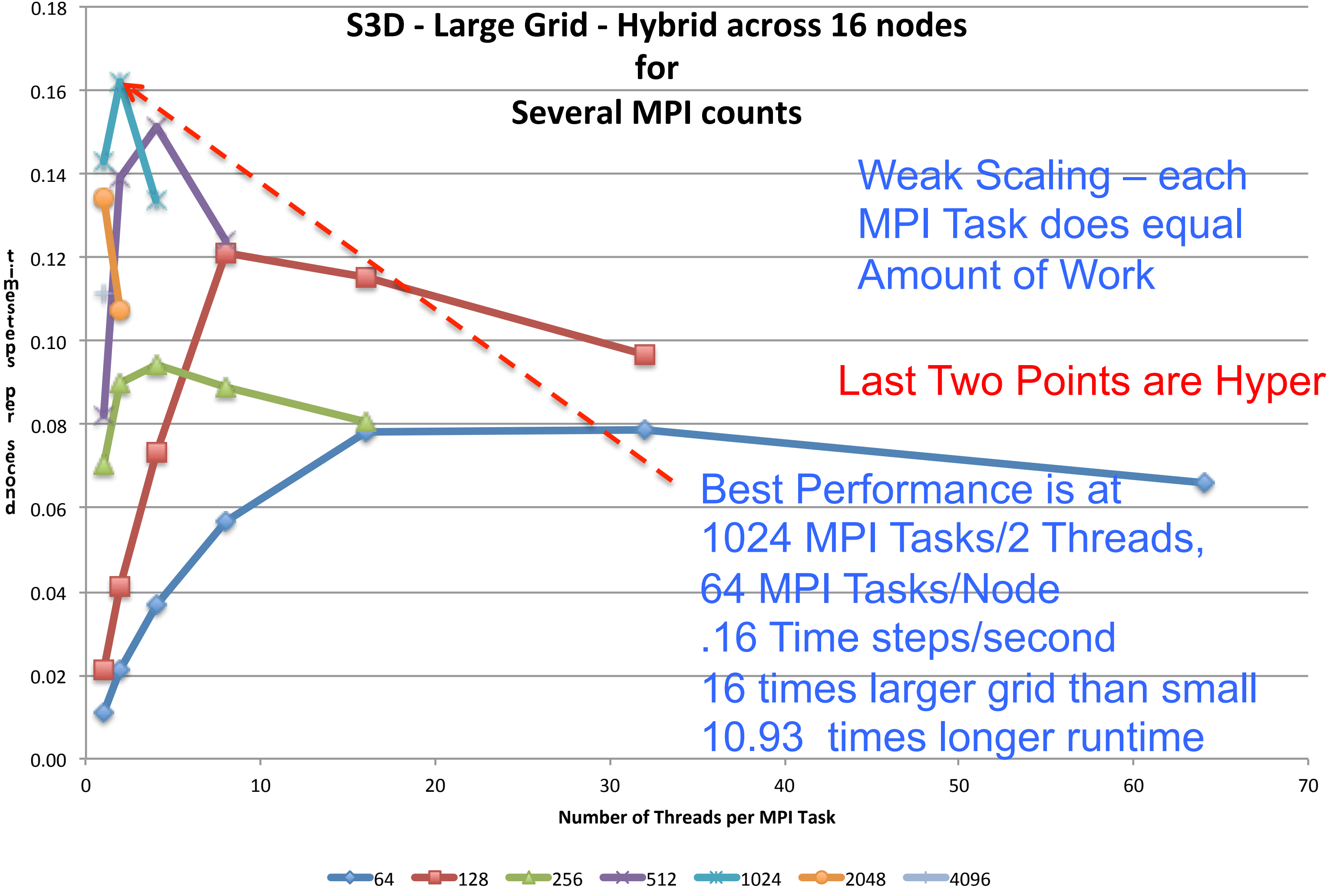


Weak Scaling – each MPI Task does equal Amount of Work

Last Two Points are Hyper-threads

Best Performance is at
512 MPI Tasks/4 Threads,
32 MPI Tasks/Node
.59 Time steps/second
4 times larger grid than small
2.96 times longer runtime

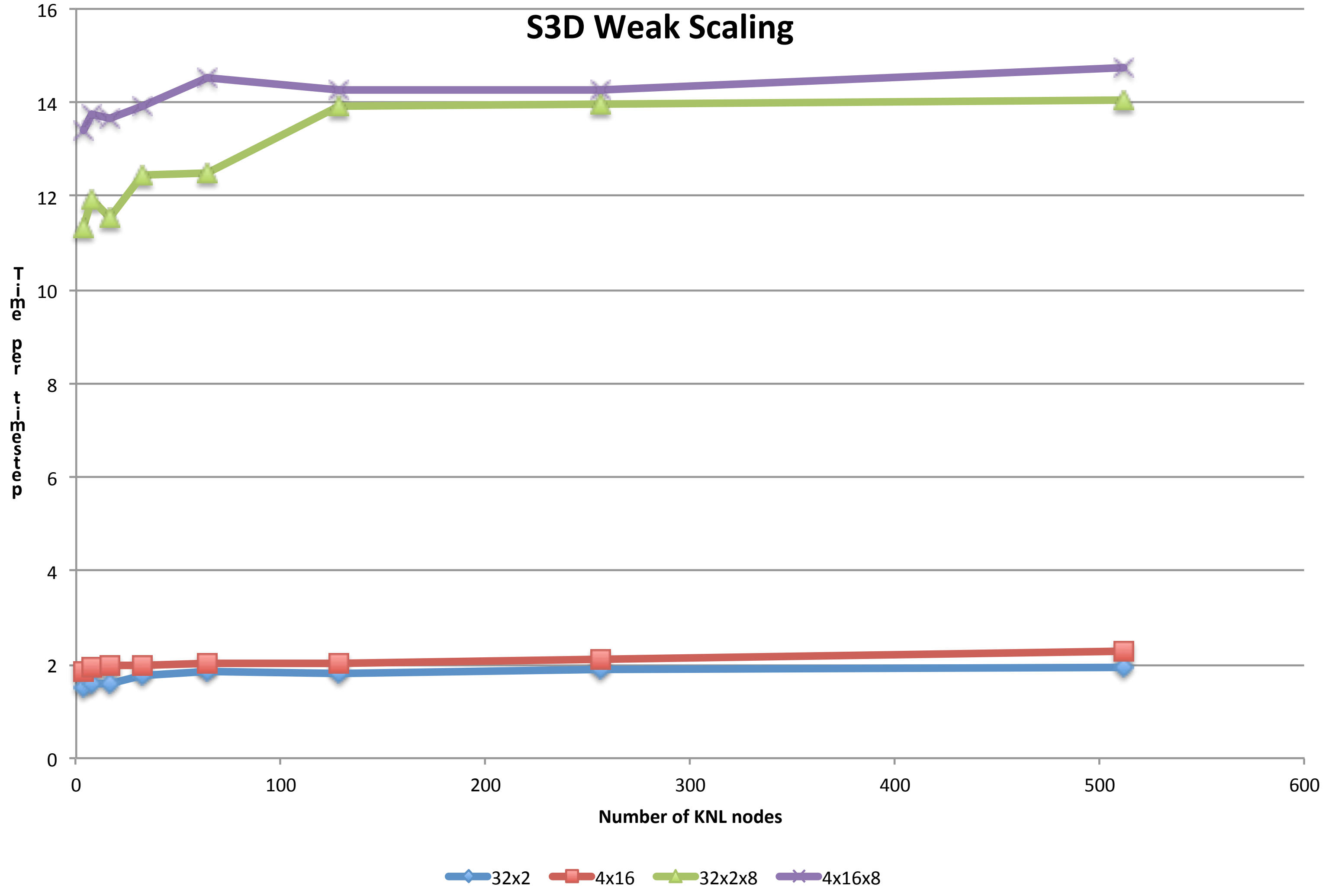
S3D - Large Grid - Hybrid across 16 nodes for Several MPI counts



Weak Scaling – each MPI Task does equal Amount of Work

Last Two Points are Hyper-threads

Best Performance is at
1024 MPI Tasks/2 Threads,
64 MPI Tasks/Node
.16 Time steps/second
16 times larger grid than small
10.93 times longer runtime


S3D Weak Scaling
Time per timestep
0
2
4
6
8
10
12
14
16
0
100
200
300
400
500
600
Number of KNL nodes
32x2
4x16
32x2x8
4x16x8

Conclusions

- **MPI works very well on KNL**
  - MPI sweet spot is always greater than 8, most times 32-64
- **Threading is extremely sensitive to NUMA features of the 2-D tile communication grid and Cache Home Agent (CHA)**
- **Advantages of using MCDRAM for cache over flat is highly dependent upon cache friendliness of application.**
  - Himeno not cache friendly - factor of 4-5
  - S3D is cache friendly – factor of 1.5-2.0

Should you use Hyper-Threads

- **Not a definite yes of no**
  - Extremely code dependent
  - HT = 2 is most likely to be best
    - Caveat – Cray doesn’t recommend running MPI across Hyper-Threads
      - Although several applications show best performance on small node counts
  - While HT helps in hiding latency, it can put increase cache pressure on all levels of cache
  - Needs to be tested

How many MPI ranks/node

- **Unless SPMD or extremely good high level OpenMP/ threading is used, at least 8 MPI ranks per node seems to be the rule with most applications getting best performance with 64 MPI ranks per node**
- **Too few MPI ranks will not be able to fully utilize the network bandwidth of number of outstanding references, leaving MPI performance on the table**
- **Be very careful with single node performance studies**
  - Must consider off-node component – especially if the ultimate goal is 1000s of nodes
  - OpenMP and MPI trade-off must be done on as many nodes as possible

OpenMP on KNL

- OpenMP is best when used within a tile
- KNL certainly has a Non-Uniform Memory Architecture
- OpenMP would be best way to employ Hyper-threads

Conclusions

- **OpenMP is a tool, not a requirement**
  - First find a sweet spot with MPI than add OpenMP
  - Do not think you need to have > 8 OpenMP threads
- **Vectorization is critical to getting good performance on KNL**
- **One must look hard at how and when to use MCDRAM**
- **No good guidance on when Hyperthreading will help**
- **KNL has a significant number of performance “knobs” that can be tuned for an individual code**
  - While lessons learned have narrowed the search space some, a significant numbers of combinations remain

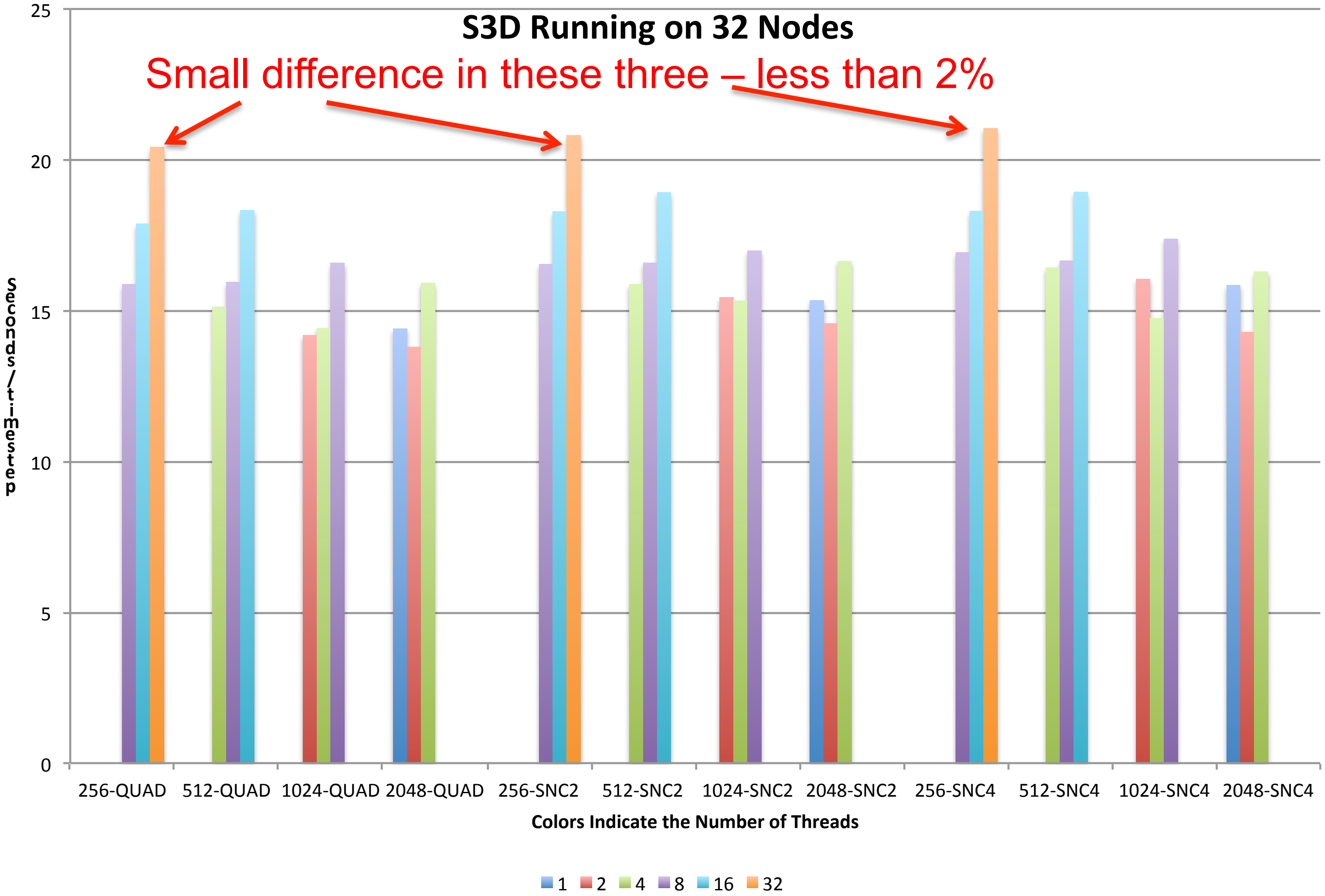
Vectorization on KNL

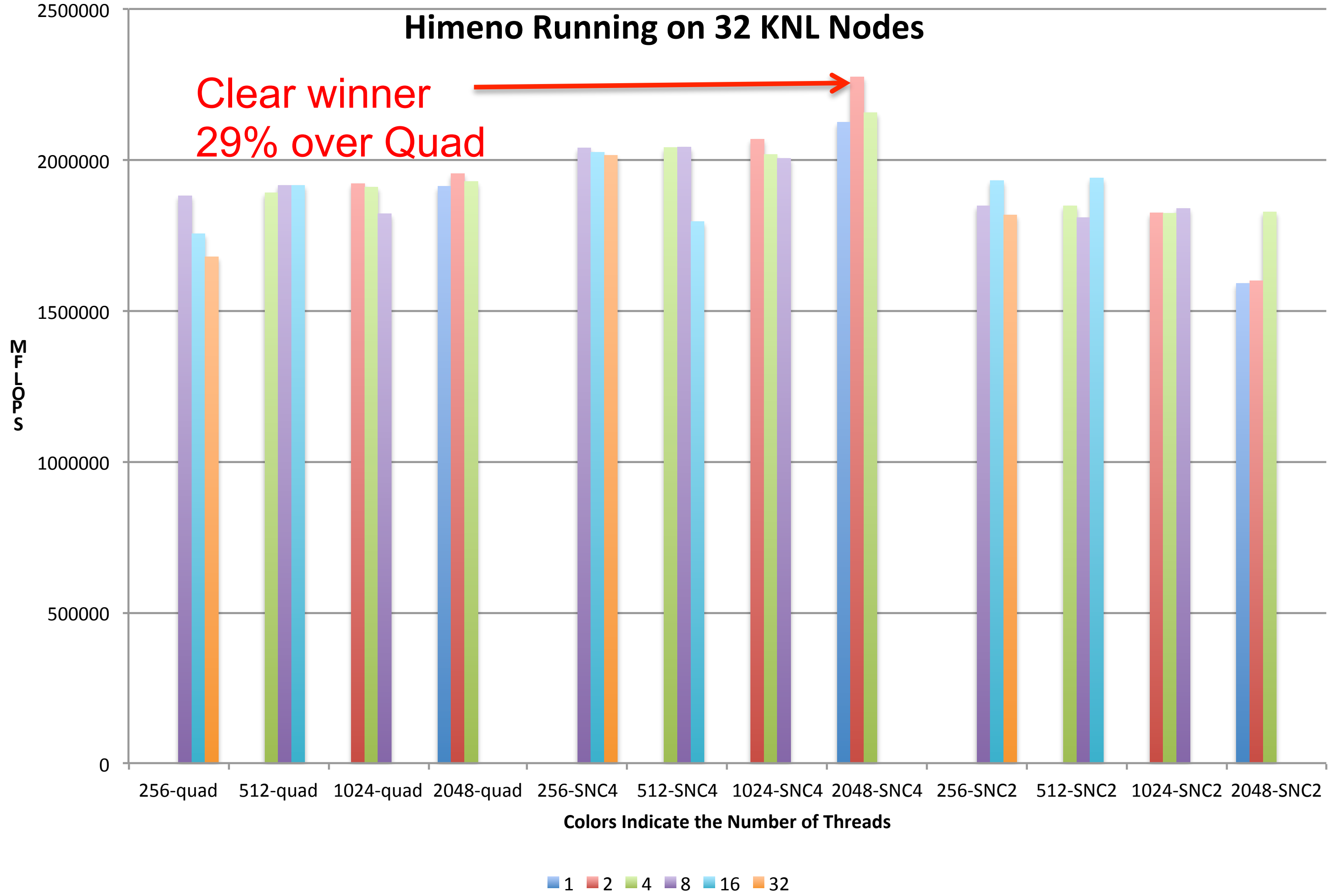
- KNL vector register length is twice that of Haswell
- KNL scalar performance is about 1/3 that of Haswell
- KNL performance is highly correlated with good vectorization
- Without good-excellent vectorization, the slow scalar performance will significantly degrade overall performance
- Vectorizing loops with lots of indirection and/or strides will probably be slower than Xeon scalar performance

MCDRAM is critical to performance

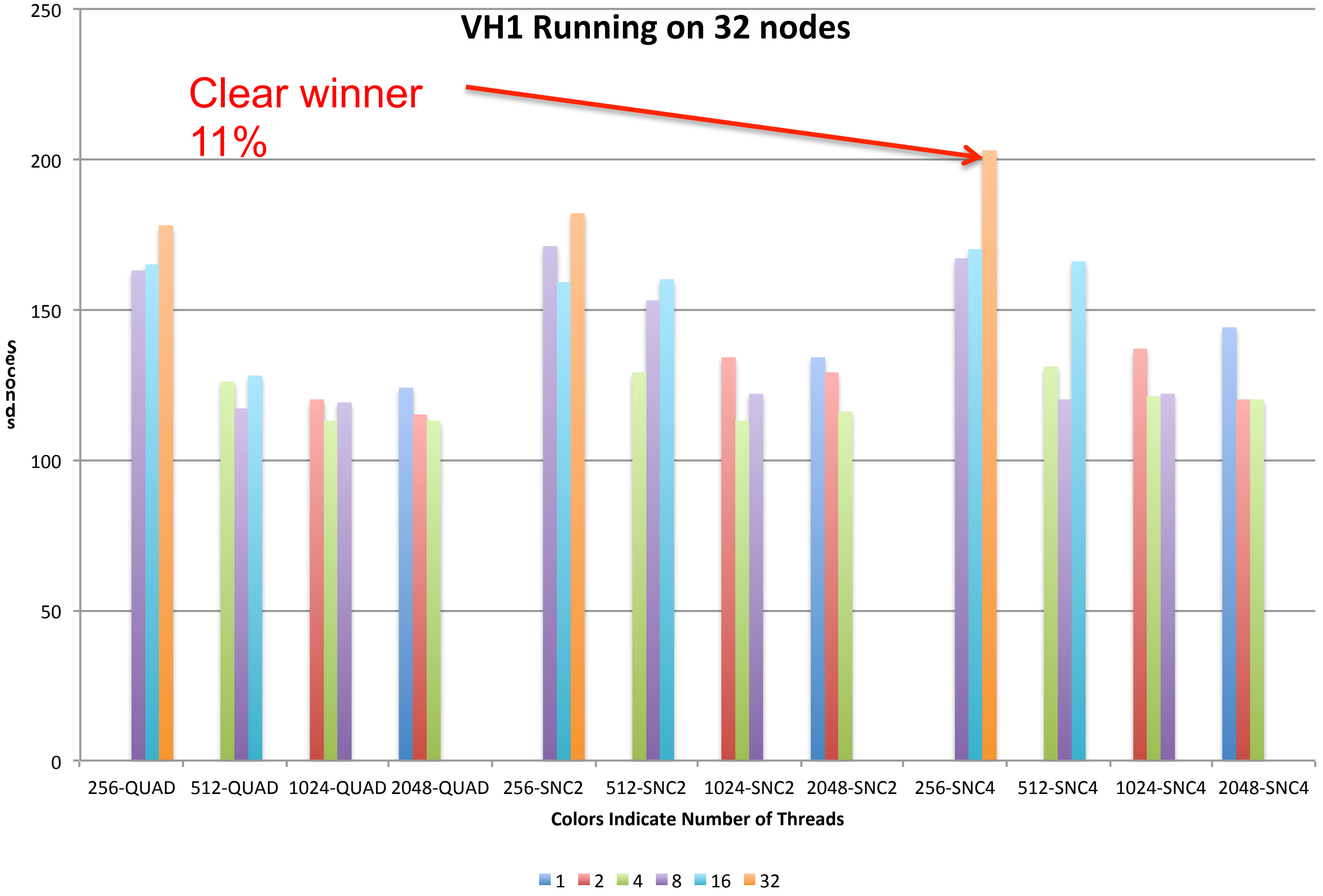
- **If an application can fit within MCDRAM, then using flat memory with numactl – membind is the best way to go.**
- **If an application cannot fit with MCDRAM, then one should consider cache, split, equal and/or flat**
  - This is extremely hard, since reserving some of MCDRAM for memory space will reduce last level cache for other variables
  - When using MCDRAM as cache, one must consider/examine issues with the direct mapped cache at scale

Small difference in these three – less than 2%



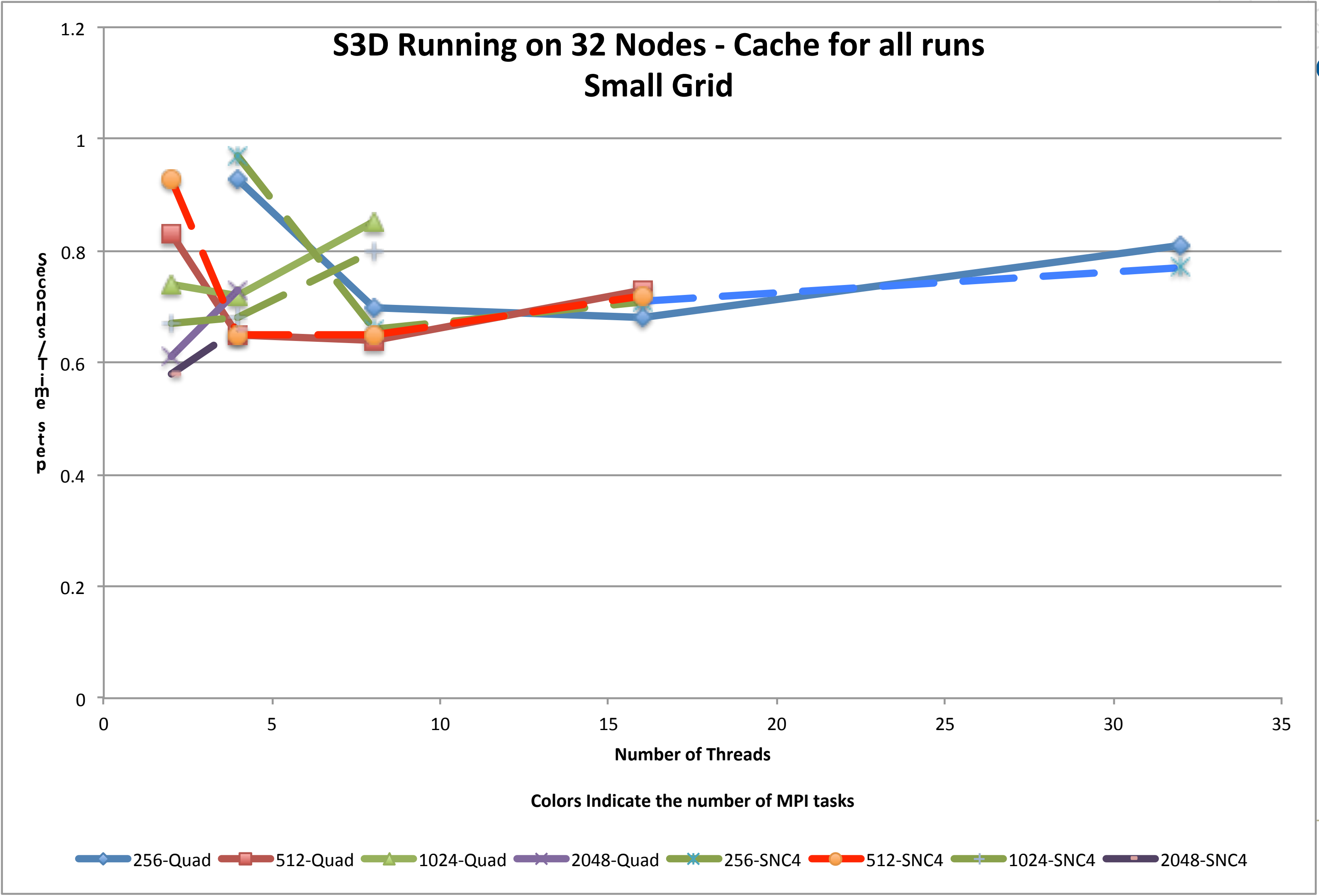


Clear winner
29% over Quad

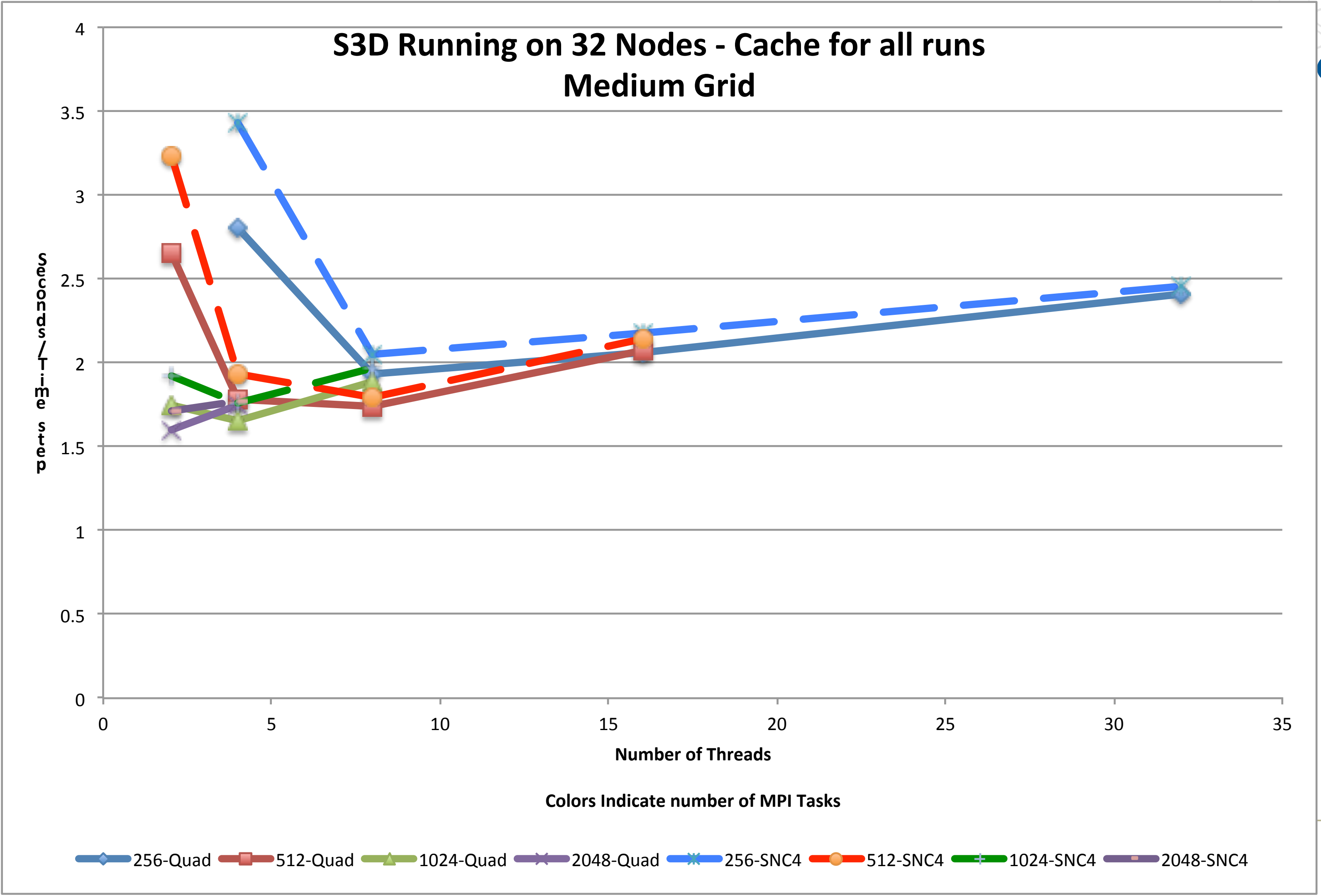


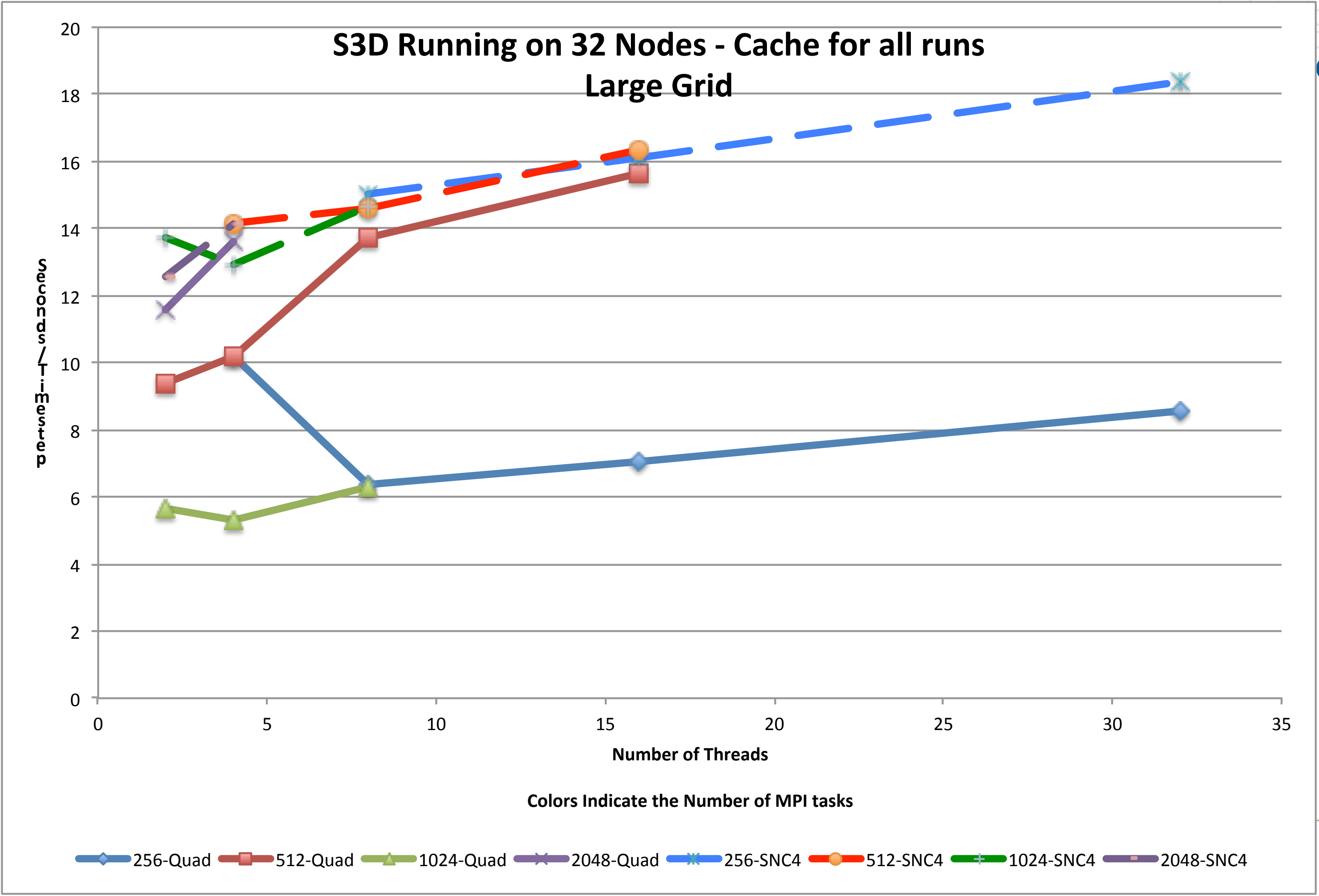
VH1 Running on 32 nodes

Clear winner
11%



S3D Running on 32 Nodes - Cache for all runs
Small Grid

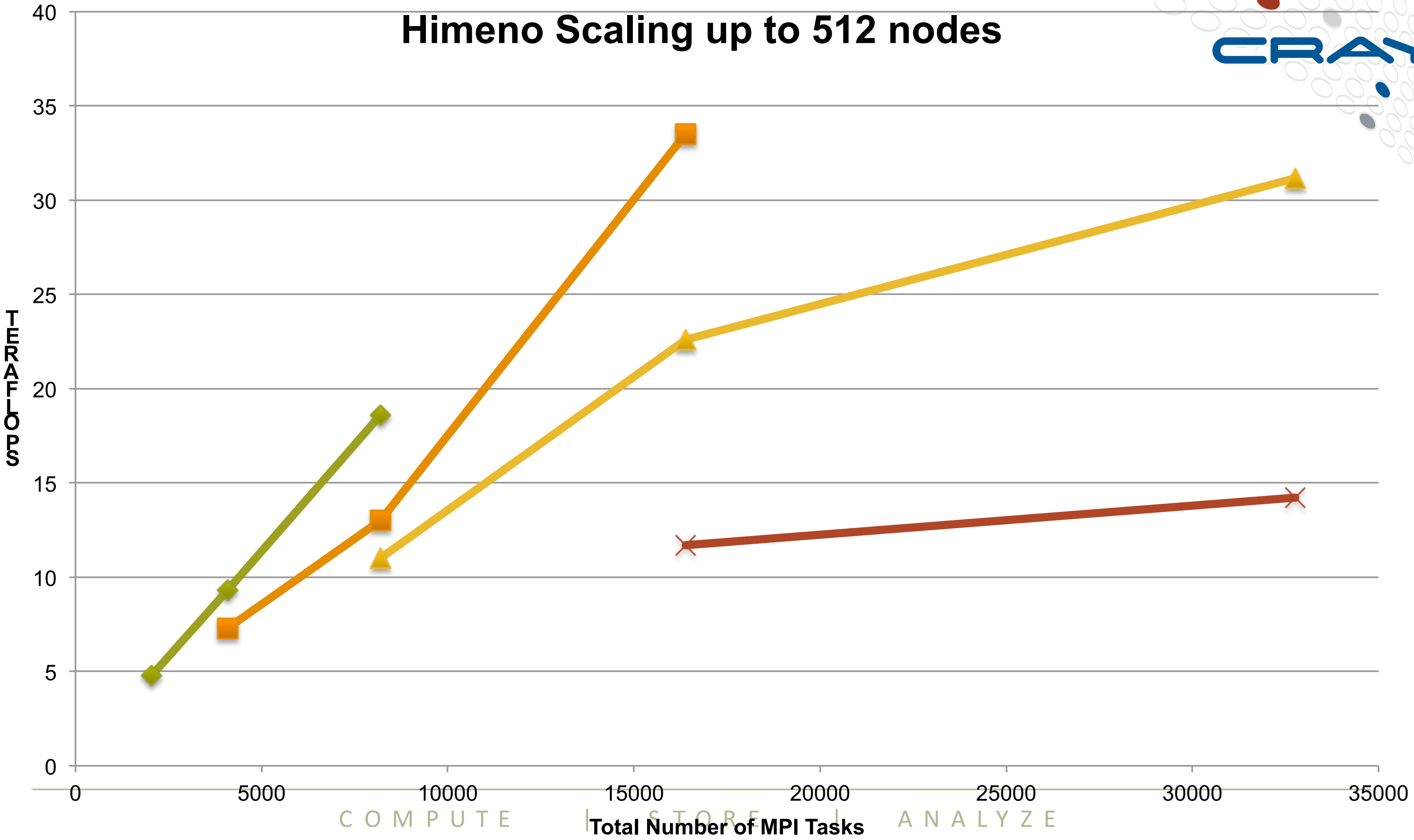

S3D Running on 32 Nodes - Cache for all runs
Medium Grid
Seconds/Time step
4
3.5
3
2.5
2
1.5
1
0.5
0
0
5
10
15
20
25
30
35
Number of Threads
Colors Indicate number of MPI Tasks
256-Quad
512-Quad
1024-Quad
2048-Quad
256-SNC4
512-SNC4
1024-SNC4
2048-SNC4

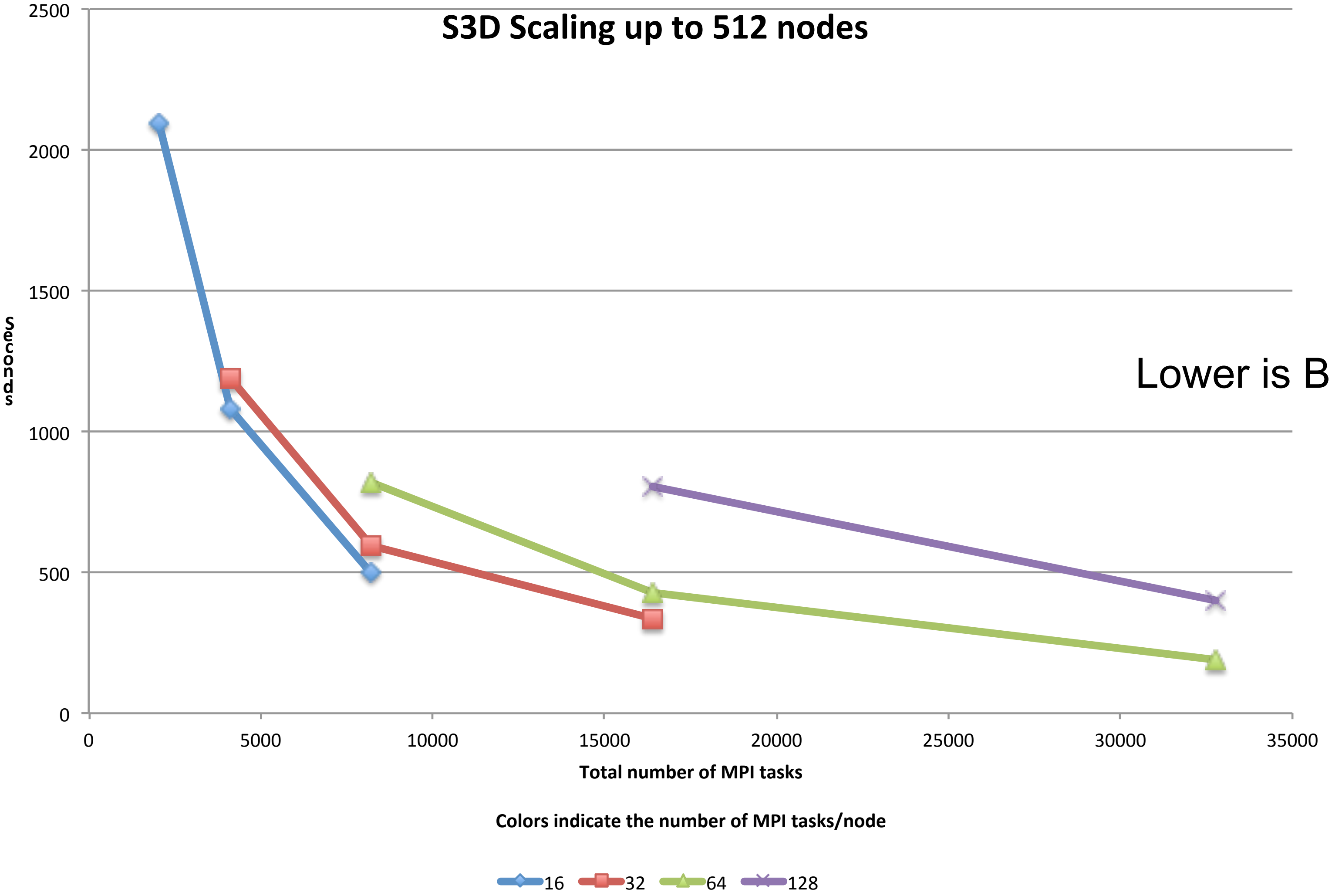

S3D Running on 32 Nodes - Cache for all runs
Large Grid
Seconds/Timestep
0
2
4
6
8
10
12
14
16
18
20
0
5
10
15
20
25
30
35
Number of Threads
Colors Indicate the Number of MPI tasks
256-Quad
512-Quad
1024-Quad
2048-Quad
256-SNC4
512-SNC4
1024-SNC4
2048-SNC4

Examine MPI scaling of existing application across all nodes, all cores

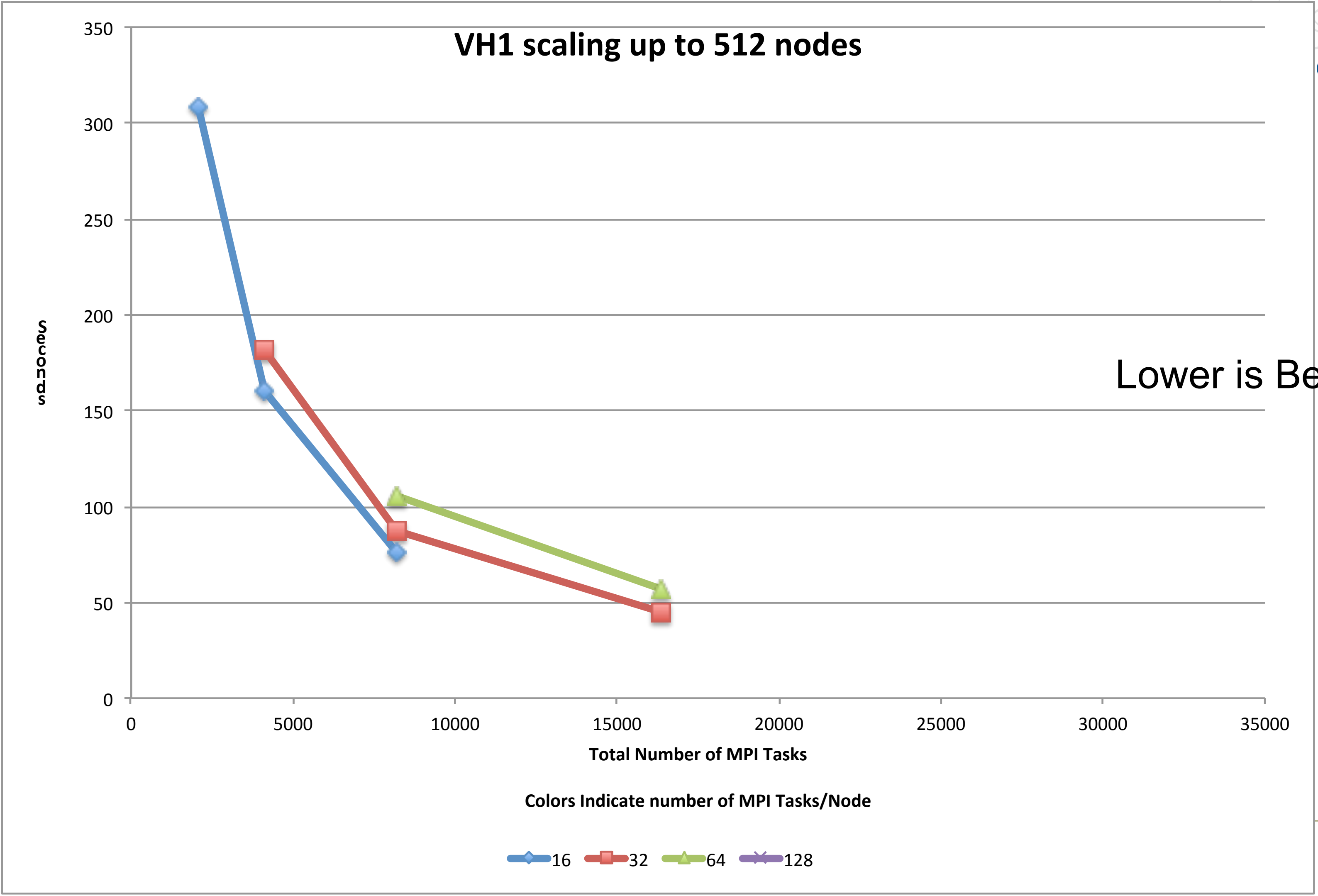
- Regardless of your current threading, examine the scaling of MPI only across all the cores of all the nodes that you would like to use.
- There will be a MPI sweet spot, the point at which MPI stops scaling.
- To date a large majority of the applications have shown that at least one MPI task/tile is the best hybrid configuration
  - 32 (34) MPI tasks – 2,4, or 8 threads
  - 64 (68) MPI tasks – 2,4 threads
- While 128 (136) MPI tasks on a nodes occasionally is faster, Cray does not recommend doing MPI on Hyper-threads

Himeno Scaling up to 512 nodes

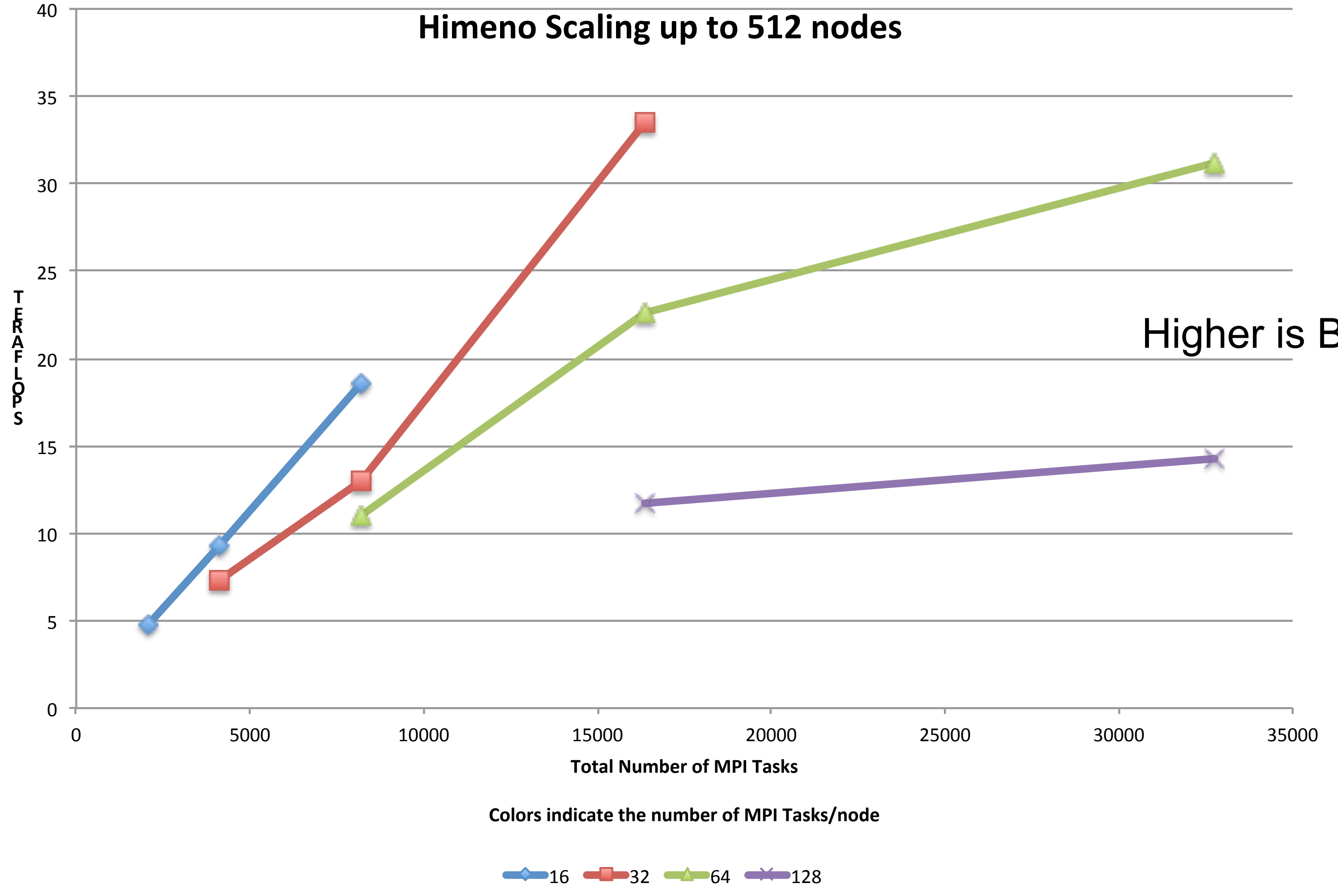




Lower is Better



Lower is Better



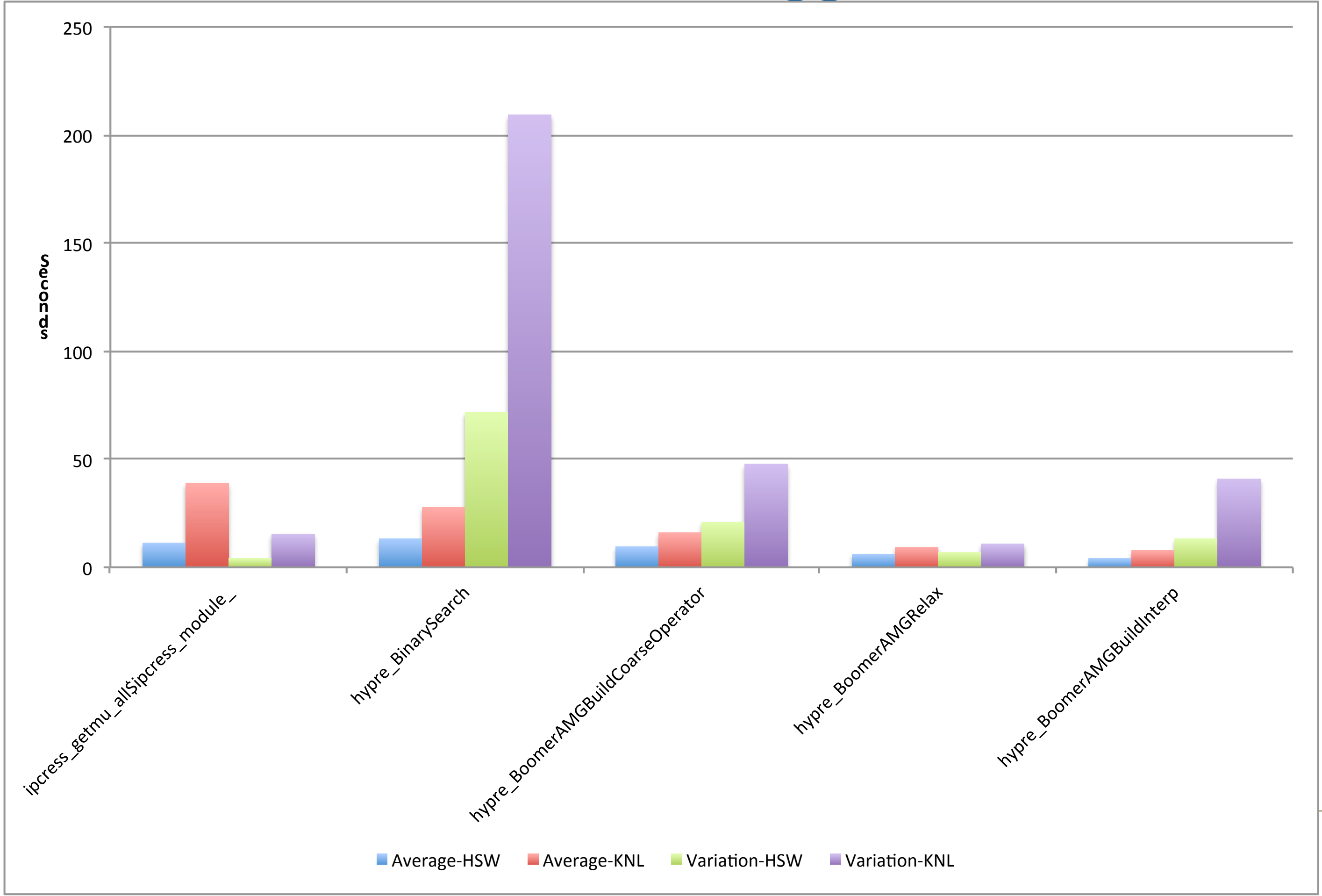
Higher is Better

Investigate load imbalance

Scalar Load Imbalance is exaggerated on KNL

```
Table 1:  Profile by Function
  Samp% |        Samp |      Imb. |     Imb. |Group
        |             |      Samp |    Samp% | Function
        |             |           |          |  PE=HIDE

 100.0% | 190,766.8 |        -- |      -- |Total
|-----------------------------------------------------------------------------
|  87.8% | 167,553.3 |        -- |      -- |MPI
||----------------------------------------------------------------------------
||  52.0% |  99,181.7 | 20,759.3 |  17.3% |MPI_Waitall
||  12.6% |  24,068.4 | 23,281.6 |  49.2% |MPI_Allgather
||  10.7% |  20,407.1 | 10,906.9 |  34.9% |MPI_Allreduce
||   3.3% |   6,245.7 |  1,399.3 |  18.3% |mpi_waitall
||   2.2% |   4,110.2 |     48.8 |   1.2% |mpi_bcast
||   2.0% |   3,795.6 | 11,579.4 |  75.4% |MPI_Isend
||   1.7% |   3,298.1 |     55.9 |   1.7% |MPI_BARRIER
||   1.2% |   2,199.5 |    639.5 |  22.5% |MPI_ALLREDUCE
||============================================================================
|   9.4% |  17,949.0 |        -- |      -- |USER
||----------------------------------------------------------------------------
||   2.0% |   3,887.4 |  1,543.6 |  28.4% |ipcress_getmu_all$ipcress_module_
||   1.5% |   2,782.6 | 20,929.4 |  88.4% |hypre_BinarySearch
||   0.8% |   1,594.7 |  4,774.3 |  75.0% |hypre_BoomerAMGBuildCoarseOperator
||   0.5% |     939.8 |  1,084.2 |  53.6% |hypre_BoomerAMGRelax
||   0.4% |     776.4 |  4,092.6 |  84.1% |hypre_BoomerAMGBuildInterp
||   0.3% |     626.9 |    101.1 |  13.9% |inside_com3$derivatives_common$derivative_module_
||   0.2% |     428.9 |    302.1 |  41.4% |hypre_CSRMatrixMatvec
||   0.2% |     391.5 |    741.5 |  65.5% |hypre_BoomerAMGCoarsen
||   0.2% |     291.3 |     59.7 |  17.0% |derivatives_common$derivative_module_
||   0.1% |     281.3 |     80.7 |  22.3% |teos_iter_3$teos_module_
||   0.0% |       0.0 |      1.0 | 100.0% |hypre_BoomerAMGSetThreshold
||============================================================================
|   2.5% |   4,807.5 |        -- |      -- |ETC
||----------------------------------------------------------------------------
||   0.7% |   1,245.7 |    493.3 |  28.4% |_EXP_Z
||   0.4% |     760.1 |    118.9 |  13.5% |_RTON
|-----------------------------------------------------------------------------
```

Scalar Load Imbalance is exaggerated on KNL


seconds
ipcress_getmu_all$ipcress_module_
hypre_BinarySearch
hypre_BoomerAMGBuildCoarseOperator
hypre_BoomerAMGRelax
hypre_BoomerAMGBuildInterp
Average-HSW
Average-KNL
Variation-HSW
Variation-KNL