- Software Optimization
- Code Modernization
- Unleash Si perf.

Rama Malladi

Intel, Bangalore

Legal Disclaimer & Optimization Notice

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

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © 2017, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

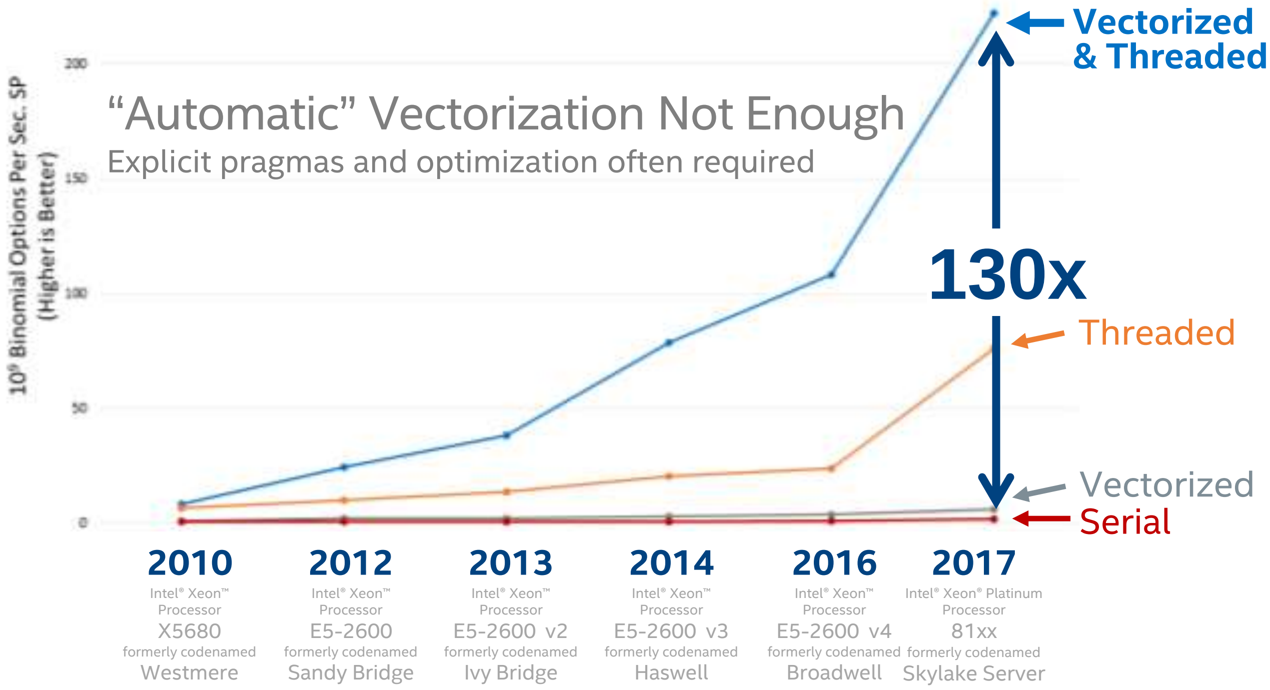
Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Vectorize & Thread or Performance Dies

Threaded + Vectorized can be much faster than either one alone



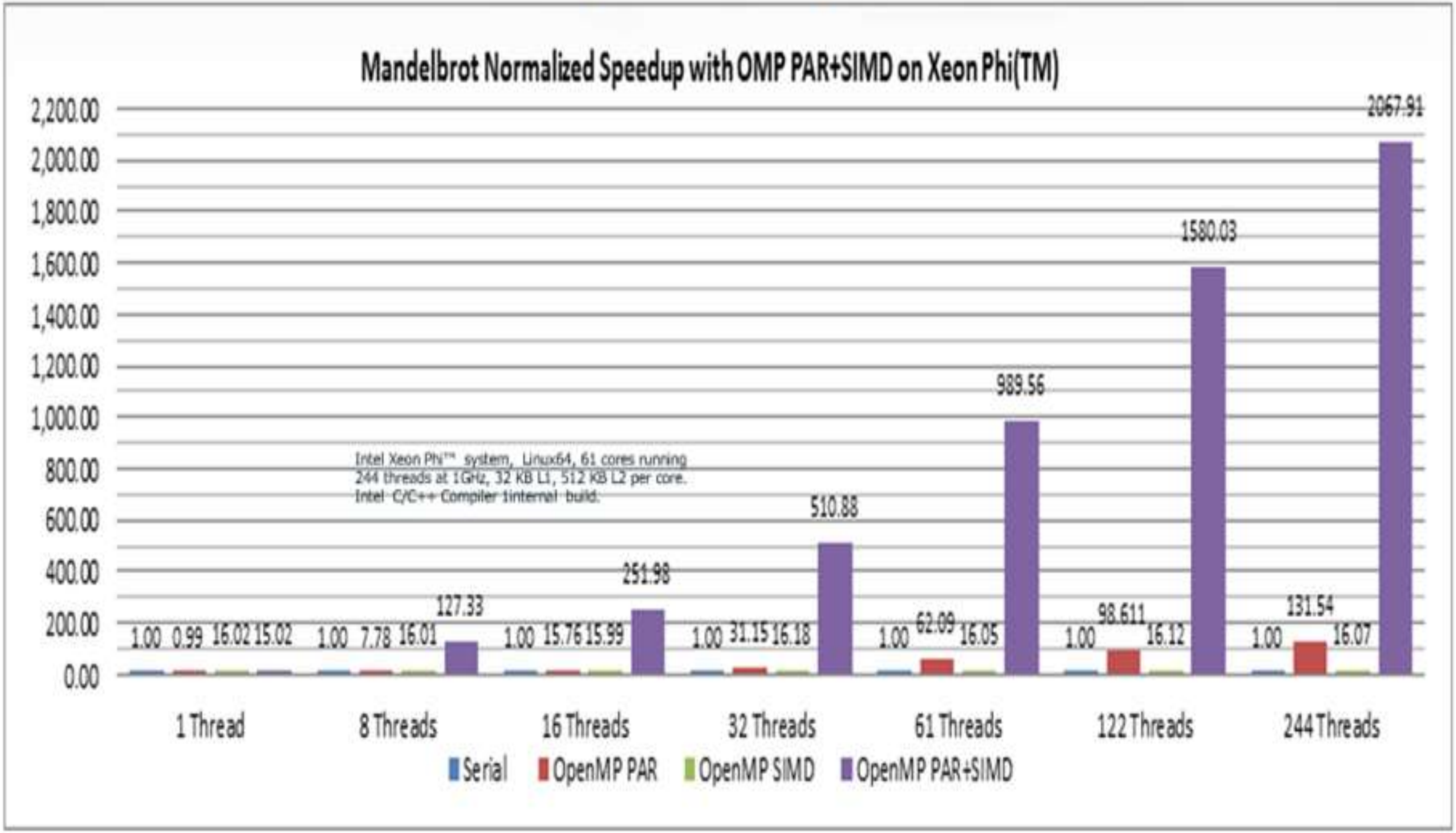
“Automatic” Vectorization Not Enough

Explicit pragmas and optimization often required

The Difference Is Growing With Each New Generation of Hardware

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance

Mandelbrot: Speedup on Xeon Phi™



Intel Xeon Phi™ system, Linux64, 61 cores running 244 threads at 1GHz, 32 KB L1, 512 KB L2 per core. Intel C/C++ Compiler 1internal build.

Optimized & Modernized Implementation

- ✓ Loop Unrolling (#pragma unroll)
  - Short loop hurts instruction scheduling.
- ✓ Threading (#pragma omp parallel)
  - Embarrassingly parallel.
  - No write conflicts and small working set.
- ✓ Vectorization (#pragma omp simd)
  - v0/v1 must be reduced.
  - max() call introduces control divergence.
  - m_r[p] should be aligned.
- ✓ Arithmetic
  - Use native exp2() call on coprocessor.

```
#pragma omp target device(0)

#pragma omp parallel for
for(int o = 0; o < nopt; o++)

{
   const REAL_T _rt_tLN2=sqrt(T[o])*vol/M_LN2;
   const REAL_T mu_tLN2 = T[o]*mu/MLN2;
   REAL_T v0 = 0, v1 = 0, res;
   #pragma omp simd  reduction(+:v0,v1)
   aligned(m_r:64) unroll(4)
   for(int p = 0; p < npath; ++p) {
     res = max(0, S[o]*exp2(v_rt_tLN2*m_r[p]
                  + mu_tLN2)-X[o]);
     v0 += res;
     v1 += res*res;
   }
   result    [o] += v0;
   confidence[o] += v1;
}
```

Availability of Tools?

Create Fast Code Faster with Intel® Parallel Studio XE

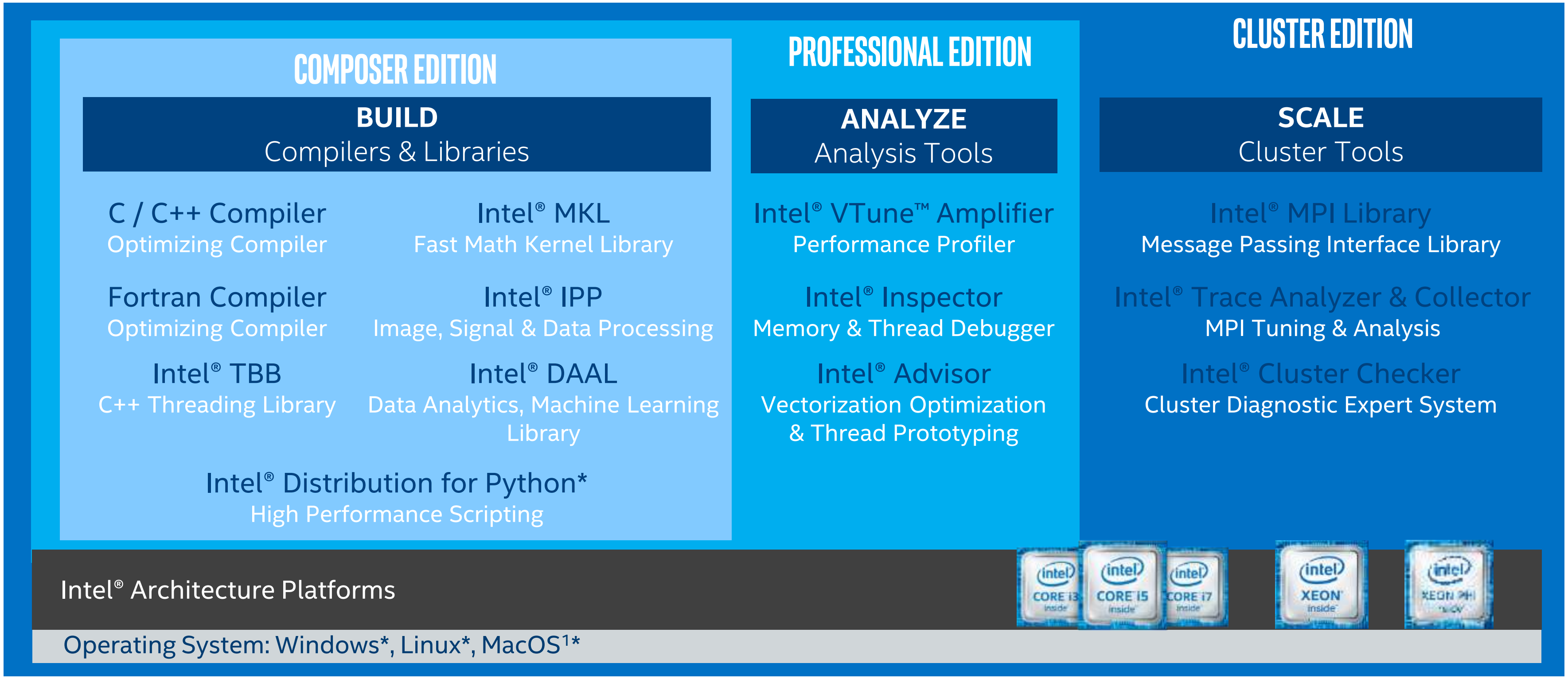
Build high performance, scalable applications for HPC, enterprise and cloud solutions running on Intel® platforms.

- Take full advantage Intel hardware and performance capabilities.
- Deliver consistent programming using **Intel® AVX-512** for Intel® Xeon® and Intel® Xeon Phi™ processors.
- **Simplify developing and modernizing code** with the latest techniques in vectorization, multi-threading, multi-node, and memory optimization.
- Use industry-leading compilers, numerical libraries, performance profilers, and code analyzers to confidently **optimize software for modern hardware**.

Applicable for **C, C++, Fortran** and **Python*** software developers. **Use standards-driven parallel models**: OpenMP*, MPI, and Intel® Threading Building Blocks.

What’s Inside Intel® Parallel Studio XE

Accelerate HPC, Enterprise & Cloud Applications



COMPOSER EDITION

BUILD
Compilers & Libraries

C / C++ Compiler
Optimizing Compiler

Intel® MKL
Fast Math Kernel Library

Fortran Compiler
Optimizing Compiler

Intel® IPP
Image, Signal & Data Processing

Intel® TBB
C++ Threading Library

Intel® DAAL
Data Analytics, Machine Learning Library

Intel® Distribution for Python*
High Performance Scripting

PROFESSIONAL EDITION

ANALYZE
Analysis Tools

Intel® VTune™ Amplifier
Performance Profiler

Intel® Inspector
Memory & Thread Debugger

Intel® Advisor
Vectorization Optimization & Thread Prototyping

CLUSTER EDITION

SCALE
Cluster Tools

Intel® MPI Library
Message Passing Interface Library

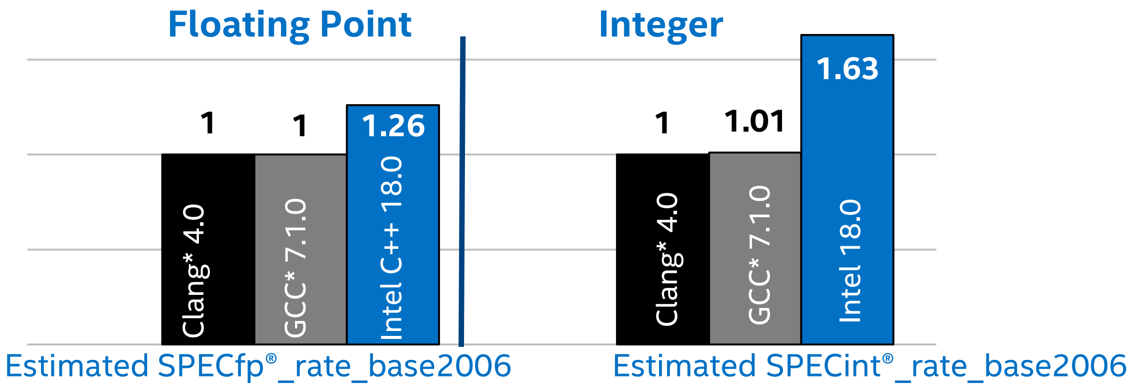
Intel® Trace Analyzer & Collector
MPI Tuning & Analysis

Intel® Cluster Checker
Cluster Diagnostic Expert System

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS1*

Boost Application Performance on Linux* Using Intel® Compiler (higher is better)



Relative geomean performance, SPEC* benchmark - higher is better

Configuration: Linux hardware: 2x Intel® Xeon® Gold 6148 CPU @ 2.40GHz, 192 GB RAM, HyperThreading is on. Software: Intel compilers 18.0, GCC 7.1.0. PGI 15.10, Clang/LLVM 4.0. Linux OS: Red Hat Enterprise Linux Server release 7.2 (Maipo), kernel 3.10.0-514.el7.x86_64. SPEC* Benchmark (www.spec.org). SmartHeap 10 was used for CXX tests when measuring SPECint® benchmarks. SPECfp® tests measure C/C++ code performance only. SPECint®_rate_base_2006 compiler switches: SmartHeap 10 were used for C++ tests. Intel C/C++ compiler 18.0: -m32 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 C++ code adds option –static . GCC 7.1.0: -m32 -Ofast -flto -march=core-avx2 -mfpmath=sse -funroll-loops . Clang 4.0: -m32 -Ofast -march=core-avx2 -flto -mfpmath=sse -funroll-loops C++ code adds option –fno-fast-math . SPECfp®_rate_base_2006 compiler switches: Intel C/C++ compiler 18.0: -m64 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 -auto-p32. C code adds option –static. Intel Fortran 18.0: -m64 -xCORE-AVX512 -ipo -O3 -no-prec-div -qopt-prefetch -qopt-mem-layout-trans=3 –static. GCC 7.1.0: -m64 -Ofast -flto -march=core-avx2 -mfpmath=sse -funroll-loops. Clang 4.0: -m64 -Ofast -march=core-avx2 -flto -mfpmath=sse -funroll-loops

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance. Benchmark Source: Intel Corporation. Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

Faster Python* with Intel® Distribution for Python*

Advance Performance Closer to Native Code

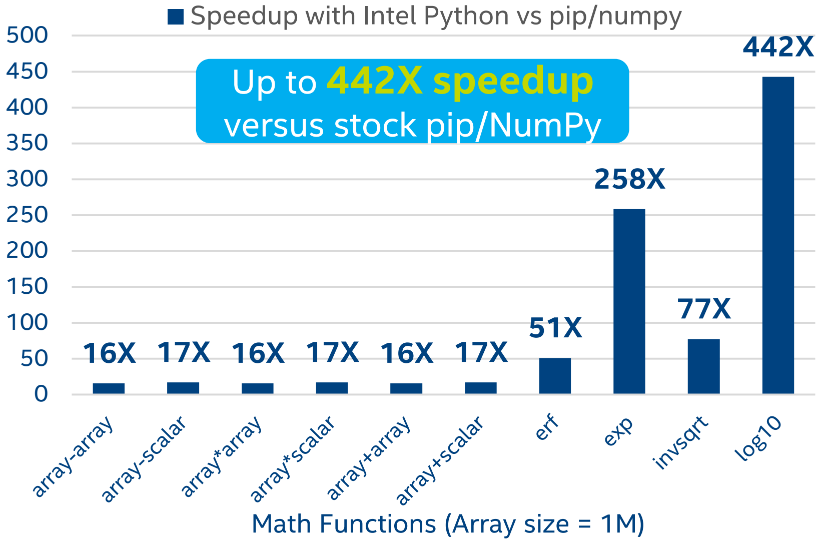
- Accelerated NumPy, SciPy, scikit-learn for scientific computing, machine learning & data analytics
- Drop-in replacement for existing Python - no code changes required
- Highly optimized for the latest Intel processors

What's New in the 2018 edition

- Updated to support Python 3.6
- Optimized scikit-learn for machine learning speedups
- Conda build recipes for custom infrastructure

Learn More: software.intel.com/distribution-for-python

Intel® Distribution for Python* Performance Speedups for Select Math Functions on Intel® Xeon™ Processors

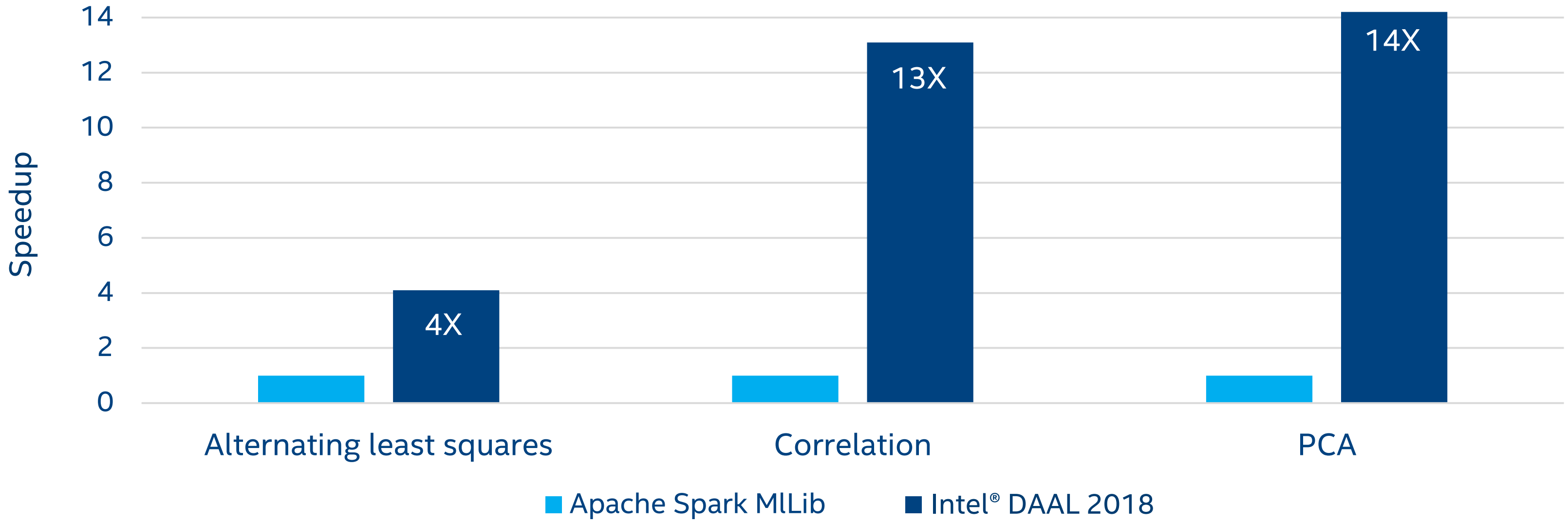


Configuration: Hardware: Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz (2 sockets, 22 cores per socket, 1 thread per core – HT is off), 256GB DDR4 @ 2400MHz. Software: Stock: CentOS Linux* release 7.3.1611 (Core), python 3.6.2, pip 9.0.1, numpy 1.13.1, scipy 0.19.1, scikit-learn 0.19.0. Intel® Distribution for Python* 2018 Gold: mkl 2018.0.0 intel_4, daal 2018.0.0.20170814, numpy 1.13.1 py36_intel_15, openmp 2018.0.0 intel_7, scipy 0.19.1 np113py36_intel_11, scikit-learn 0.18.2 np113py36_intel_3

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance. Benchmark Source: Intel Corporation.

Intel® DAAL 2018 vs Apache Spark* MlLib Performance

Intel® Data Analytics Acceleration Library (Intel® DAAL)



Configuration: 2x Intel® Xeon® E5-2660 CPU @ 2.60GHz, 128 GB, Intel® DAAL 2018; Alternating Least Squares – Users=1M Products=1M Ratings=10M Factors=100 Iterations=1 MLLib time=165.9 sec DAAL time=40.5 sec Gain=4.1x; Correlation – N=1M P=2000 size=37 GB Mllib time=169.2 sec DAAL=12.9 sec Gain=13.1x; PCA – n=10M p=1000 Partitions=360 Size=75 GB Mllib=246.6 sec DAAL (seq)=17.4 sec Gain=14.2x
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks. Source: Intel Corporation - performance measured in Intel labs by Intel employees. Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804.

Intel® Parallel Studio XE: High Performance, Scalable Software across Multiple Industries

Energy		Schlumberger 10X		
EDA		Mentor Graphics 11X		
Science & Research	*the* **Walker Molecular Dynamics** *lab* 4X	Novosibirsk State University *THE REAL SCIENCE 3X	Kyoto University 8X	NERSC 35%
Manufacturing	Altair 1.4X	esi 4X		
Government		AWE 25X		
Computer Software	FIXSTARS 2.5X	FLOW Science 1.25X		nik Software 1.3X
IT	NEC 5X	f5		OPENCASCADE 2X
Healthcare		MGH MASSACHUSETTS GENERAL HOSPITAL 20X		
Digital Media	DreamWorks			
Telecommunications		pexip 2.5X		

Software & workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark & MobileMark, are measured using specific computer systems, components, software, operations & functions. Any change to any of those factors may cause the results to vary. You should consult other information & performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance.

Extracting Si Performance

Roofline Analysis?

How to determine if we got the best / peak performance?

- Run GEMM?
- LINPACK?
- STREAM bandwidth tests?
- Latency benchmarks?
- …
- Get theoretical peak possible for the code?

Roofline Analysis

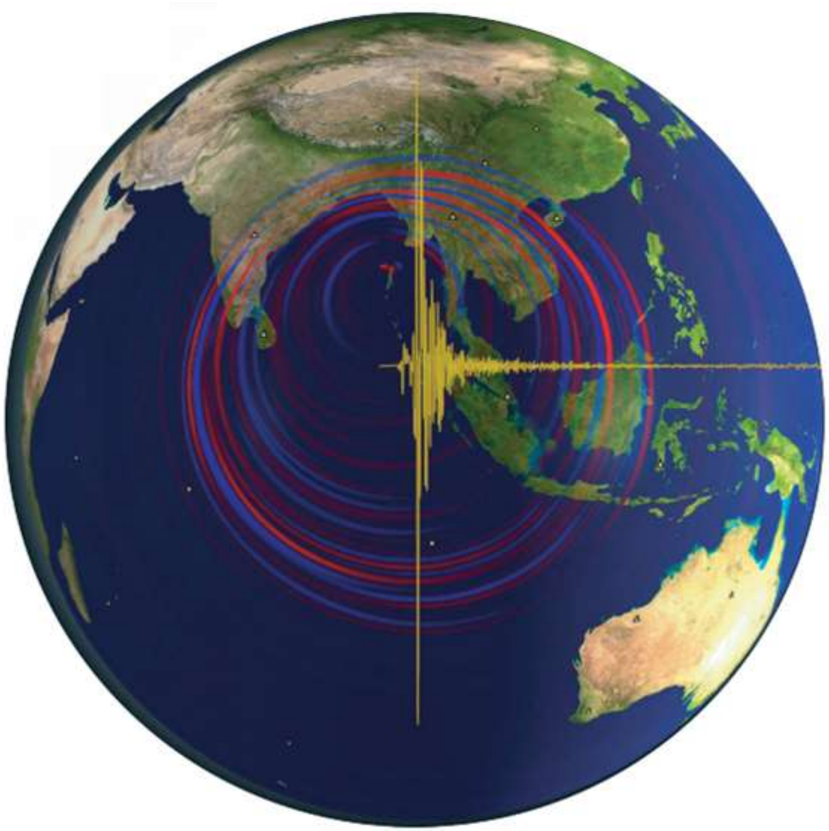
“paper-pen” exercise

```
float *A, *B, *C, d;
for(i=0; i<n; i++)
{
  A[i] = B[i] + d * C[i];
}
```

The above code on Intel Xeon Phi is bound by:

- Compute?
- Bandwidth?

What is specfem3D_globe?



The software package SPECFEM3D_GLOBE simulates three-dimensional global and regional seismic wave propagation based upon the spectral-element method (SEM).

A time-step algorithm which simulates the propagation of earth waves given the initial conditions, mesh coordinates/ details of the earth crust.

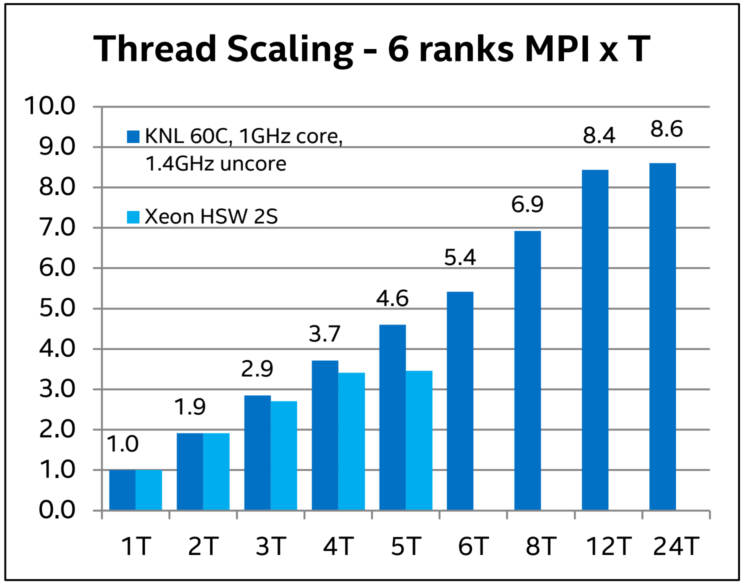
Performance of the code is measured by the time taken to simulate “n” time-steps for a given mesh volume. Typically focus is to get whole earth simulations instead of partial earth crust/ regions.

More details:
http://geodynamics.org/cig/software/specfem3d_globe/

Basic Performance Analysis

Understand application behavior out-of-the-box

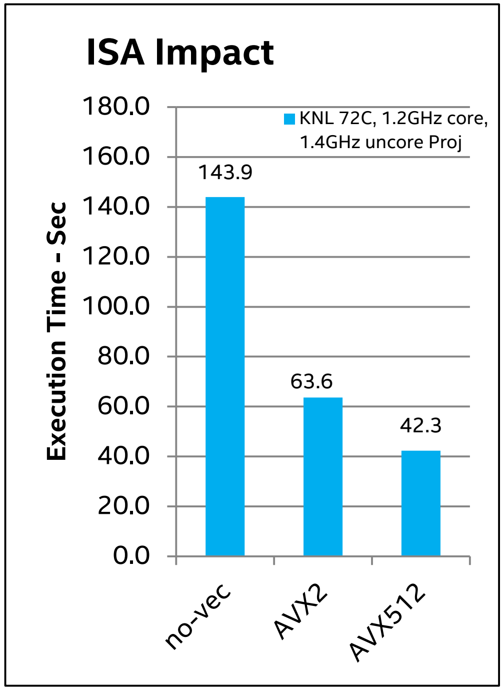
Thread Scaling



Good thread scaling

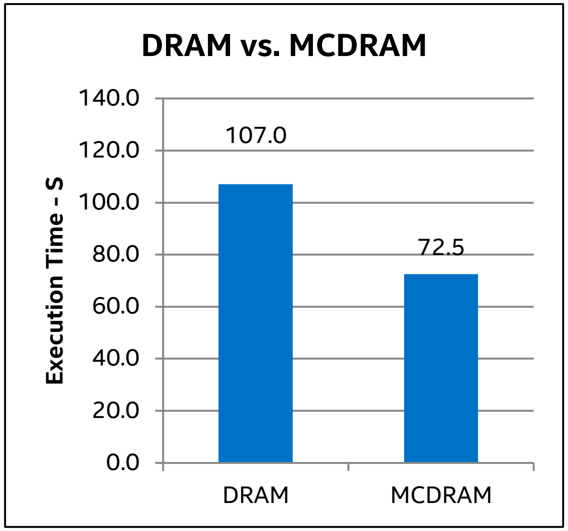
Performance relative to 1T

Instruction Set



30% Gain w KNL AVX512

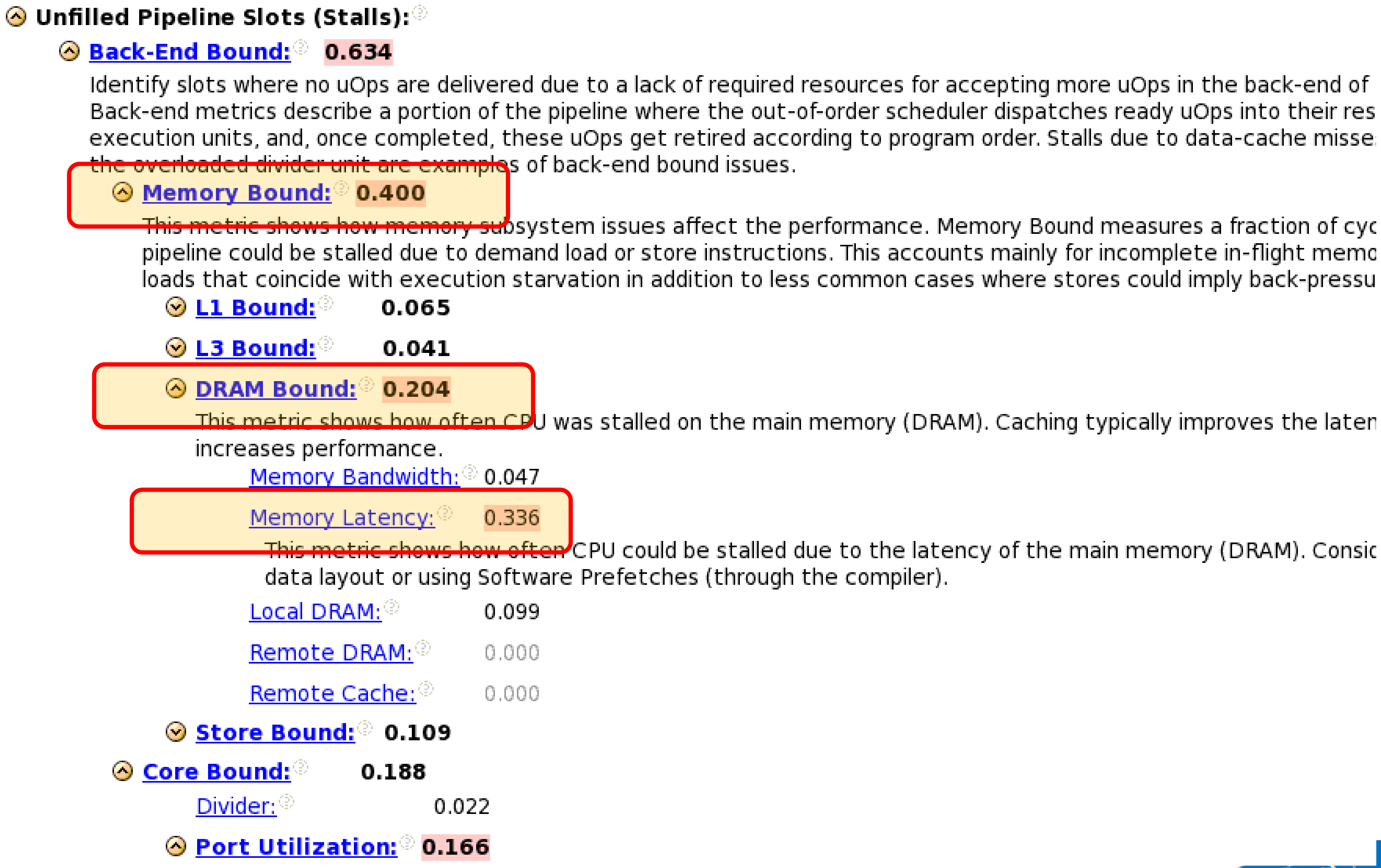
MCDRAM vs. DRAM - KNL



~1.3x gain using high bandwidth memory

VTune: *General Exploration*

Memory Latency Issues



Unfilled Pipeline Slots (Stalls):

Back-End Bound: **0.634**

Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of
Back-end metrics describe a portion of the pipeline where the out-of-order scheduler dispatches ready uOps into their res
execution units, and, once completed, these uOps get retired according to program order. Stalls due to data-cache misse
the overloaded divider unit are examples of back-end bound issues.

Memory Bound: **0.400**

This metric shows how memory subsystem issues affect the performance. Memory Bound measures a fraction of cyc
pipeline could be stalled due to demand load or store instructions. This accounts mainly for incomplete in-flight memo
loads that coincide with execution starvation in addition to less common cases where stores could imply back-pressu

L1 Bound: **0.065**

L3 Bound: **0.041**

DRAM Bound: **0.204**

This metric shows how often CPU was stalled on the main memory (DRAM). Caching typically improves the laten
increases performance.

Memory Bandwidth: 0.047

Memory Latency: 0.336

This metric shows how often CPU could be stalled due to the latency of the main memory (DRAM). Consic
data layout or using Software Prefetches (through the compiler).

Local DRAM: 0.099

Remote DRAM: 0.000

Remote Cache: 0.000

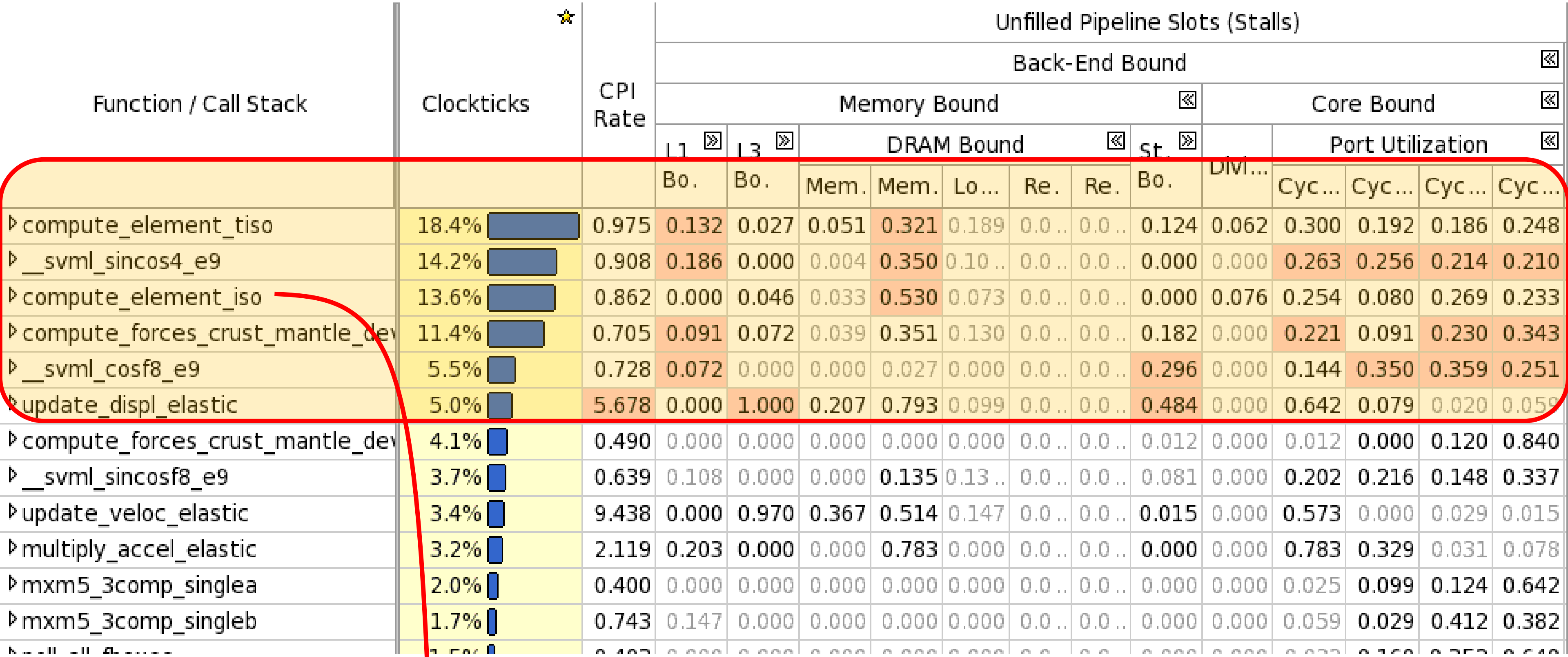
Store Bound: **0.109**

Core Bound: **0.188**

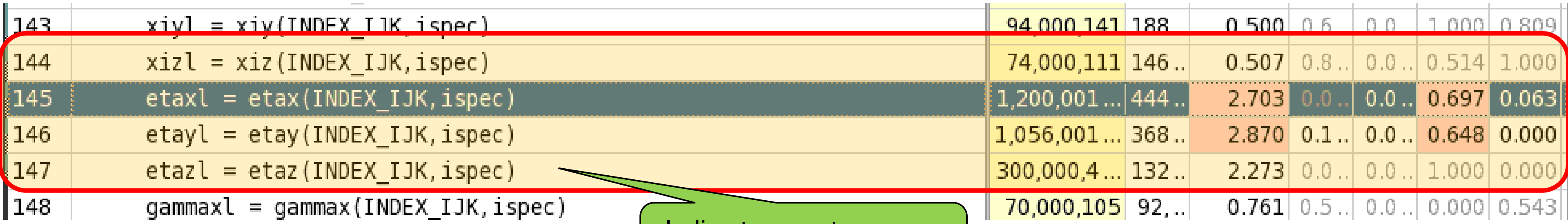
Divider: 0.022

Port Utilization: **0.166**

Top hotspots are *Memory bound*



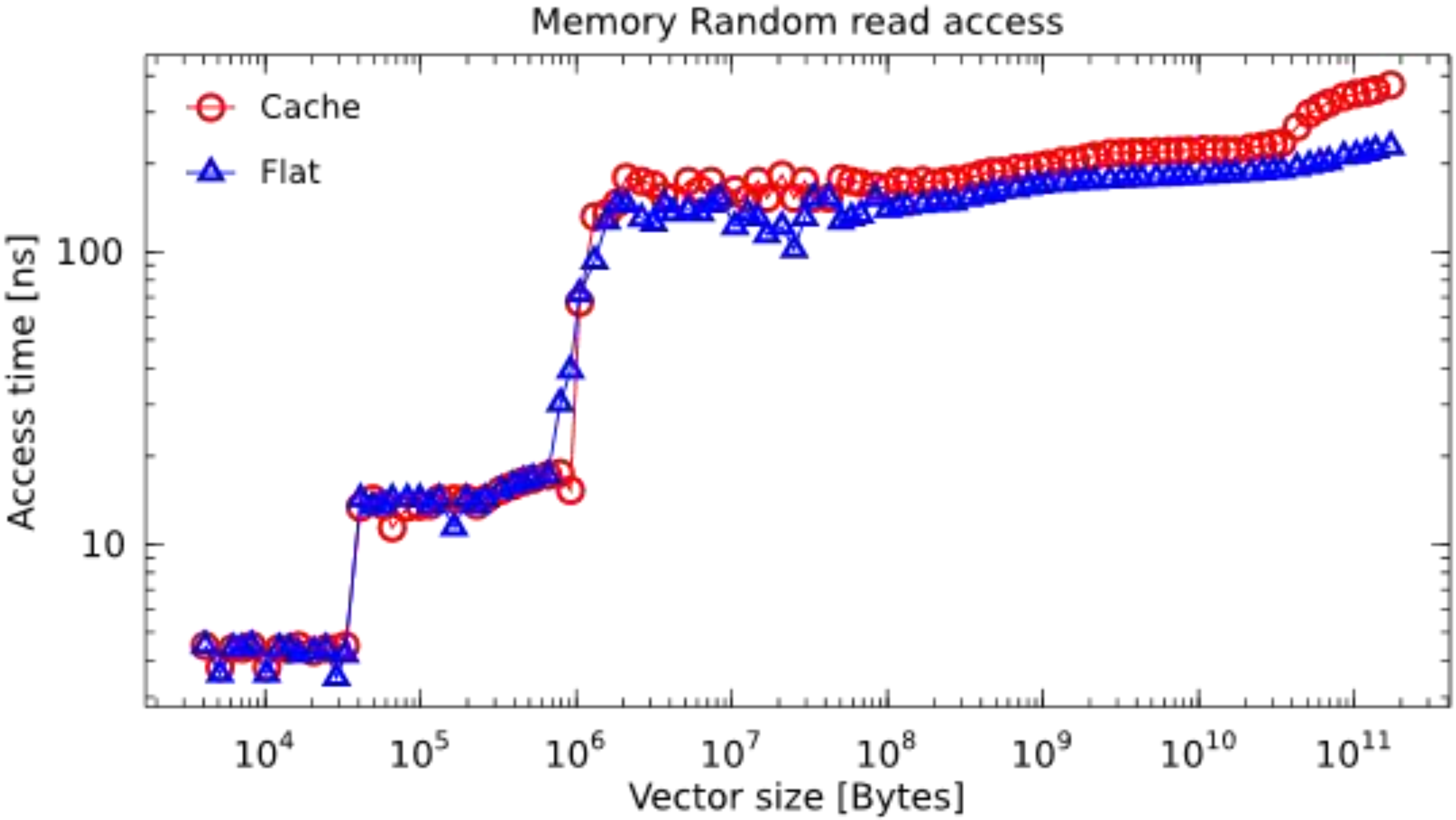
			Unfilled Pipeline Slots (Stalls)												
			Back-End Bound												
Function / Call Stack	Clockticks	CPI Rate	Memory Bound								Core Bound				
			L1	L3	DRAM Bound					St.	DIVI...	Port Utilization			
			Bo.	Bo.	Mem.	Mem.	Lo...	Re.	Re.	Bo.		Cyc...	Cyc...	Cyc...	Cyc...
compute_element_tiso	18.4%	0.975	0.132	0.027	0.051	0.321	0.189	0.0..	0.0..	0.124	0.062	0.300	0.192	0.186	0.248
__svml_sincos4_e9	14.2%	0.908	0.186	0.000	0.004	0.350	0.10..	0.0..	0.0..	0.000	0.000	0.263	0.256	0.214	0.210
compute_element_iso	13.6%	0.862	0.000	0.046	0.033	0.530	0.073	0.0..	0.0..	0.000	0.076	0.254	0.080	0.269	0.233
compute_forces_crust_mantle_dev	11.4%	0.705	0.091	0.072	0.039	0.351	0.130	0.0..	0.0..	0.182	0.000	0.221	0.091	0.230	0.343
__svml_cosf8_e9	5.5%	0.728	0.072	0.000	0.000	0.027	0.000	0.0..	0.0..	0.296	0.000	0.144	0.350	0.359	0.251
update_displ_elastic	5.0%	5.678	0.000	1.000	0.207	0.793	0.099	0.0..	0.0..	0.484	0.000	0.642	0.079	0.020	0.059
compute_forces_crust_mantle_dev	4.1%	0.490	0.000	0.000	0.000	0.000	0.000	0.0..	0.0..	0.012	0.000	0.012	0.000	0.120	0.840
__svml_sincosf8_e9	3.7%	0.639	0.108	0.000	0.000	0.135	0.13..	0.0..	0.0..	0.081	0.000	0.202	0.216	0.148	0.337
update_veloc_elastic	3.4%	9.438	0.000	0.970	0.367	0.514	0.147	0.0..	0.0..	0.015	0.000	0.573	0.000	0.029	0.015
multiply_accel_elastic	3.2%	2.119	0.203	0.000	0.000	0.783	0.000	0.0..	0.0..	0.000	0.000	0.783	0.329	0.031	0.078
mxm5_3comp_singlea	2.0%	0.400	0.000	0.000	0.000	0.000	0.000	0.0..	0.0..	0.000	0.000	0.025	0.099	0.124	0.642
mxm5_3comp_singleb	1.7%	0.743	0.147	0.000	0.000	0.000	0.000	0.0..	0.0..	0.000	0.000	0.059	0.029	0.412	0.382



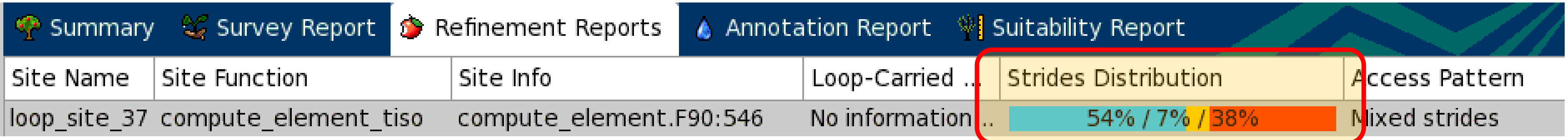
143	xiyl = xiy(INDEX_IJK,ispec)	94,000,141	188..	0.500	0.6..	0.0..	1.000	0.809
144	xizl = xiz(INDEX_IJK,ispec)	74,000,111	146..	0.507	0.8..	0.0..	0.514	1.000
145	etaxl = etax(INDEX_IJK,ispec)	1,200,001...	444..	2.703	0.0..	0.0..	0.697	0.063
146	etayl = etay(INDEX_IJK,ispec)	1,056,001...	368..	2.870	0.1..	0.0..	0.648	0.000
147	etazl = etaz(INDEX_IJK,ispec)	300,000,4...	132..	2.273	0.0..	0.0..	1.000	0.000
148	gammaxl = gammax(INDEX_IJK,ispec)	70,000,105	92,..	0.761	0.5..	0.0..	0.000	0.543

Indirect access to arrays… with "ispec" as index

Random Access Latency

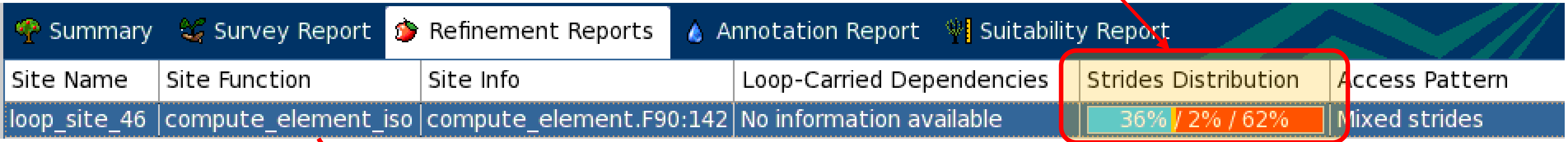


Vector Advisor

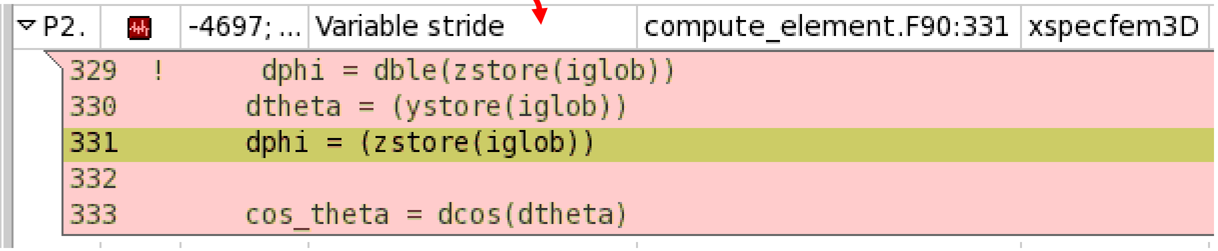


Site Name	Site Function	Site Info	Loop-Carried ...	Strides Distribution	Access Pattern
loop_site_37	compute_element_tiso	compute_element.F90:546	No information ..	54% / 7% / 38%	Mixed strides

Mix of *unit*, *fixed* and *random* stride access…



Site Name	Site Function	Site Info	Loop-Carried Dependencies	Strides Distribution	Access Pattern
loop_site_46	compute_element_iso	compute_element.F90:142	No information available	36% / 2% / 62%	Mixed strides



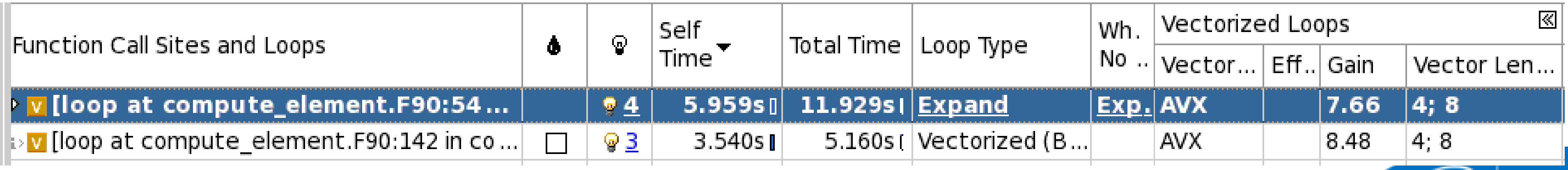
P2.		-4697; ...	Variable stride	compute_element.F90:331	xspecfem3D

```
!       dphi = dble(zstore(iglob))
       dtheta = (ystore(iglob))
       dphi = (zstore(iglob))

       cos_theta = dcos(dtheta)
```

'random' stride:
`iglob = ibool(INDEX_IJK,ispec)`

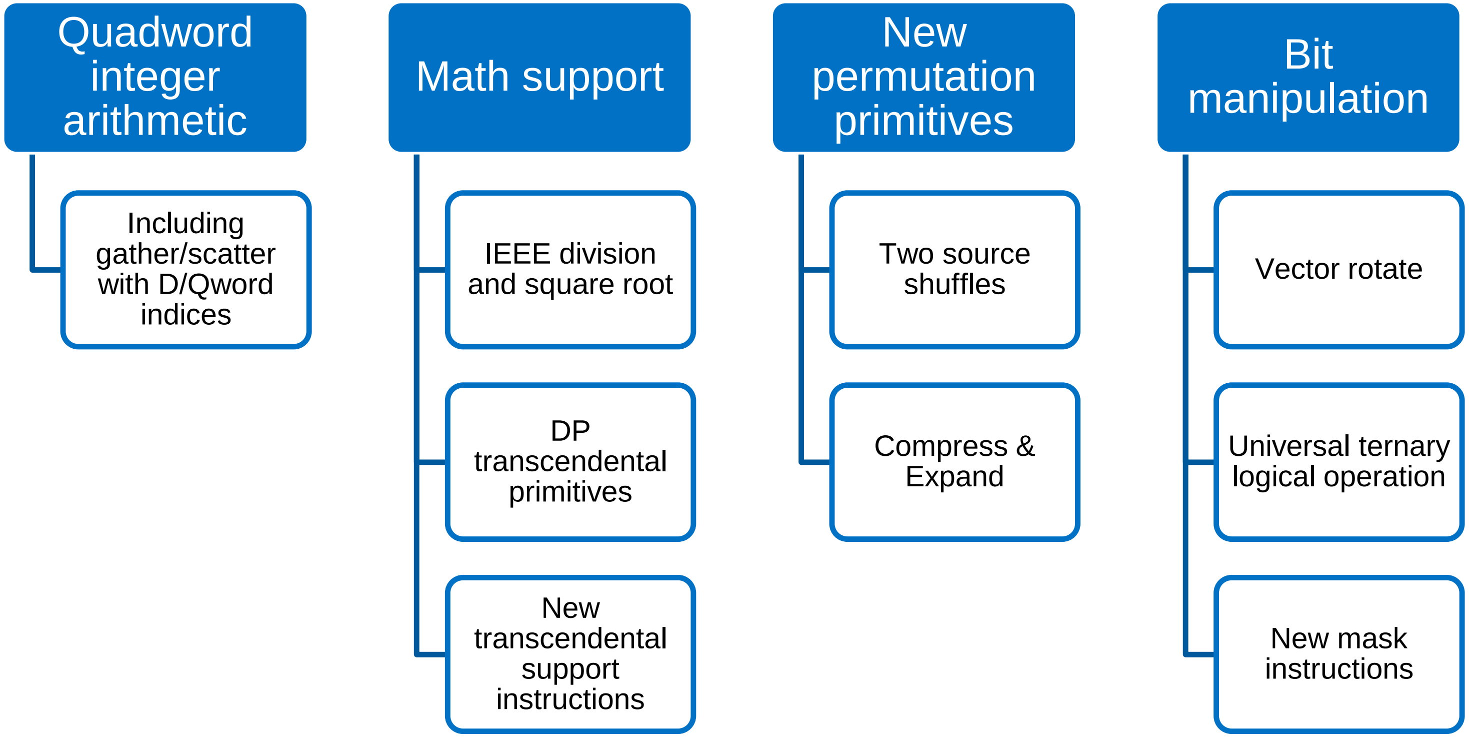
Hotspot loops are Vectorized



Function Call Sites and Loops			Self Time	Total Time	Loop Type	Wh. No ..	Vectorized Loops			
							Vector...	Eff..	Gain	Vector Len...
[loop at compute_element.F90:54...		4	5.959s	11.929s	Expand	Exp.	AVX		7.66	4; 8
[loop at compute_element.F90:142 in co...	□	3	3.540s	5.160s	Vectorized (B...		AVX		8.48	4; 8

AVX-512 Designed for HPC

- Promotions of many AVX and AVX2 instructions to AVX-512
  - 32-bit and 64-bit floating-point instructions from AVX
    - Scalar and 512-bit
  - 32-bit and 64-bit integer instructions from AVX2
- Many new instructions to speedup HPC workloads



Quadword integer arithmetic
- Including gather/scatter with D/Qword indices

Math support
- IEEE division and square root
- DP transcendental primitives
- New transcendental support instructions

New permutation primitives
- Two source shuffles
- Compress & Expand

Bit manipulation
- Vector rotate
- Universal ternary logical operation
- New mask instructions

AVX-512 features (I): More & Bigger Registers

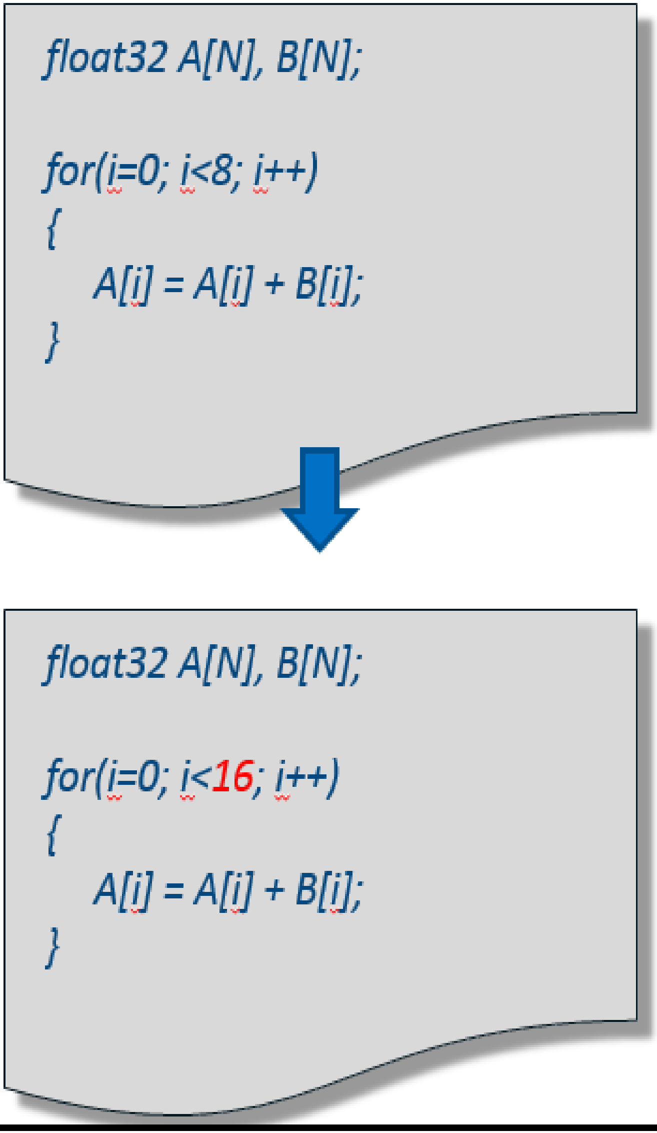
AVX: VADDPS YMM0, YMM3, [mem]

- Up to 16 AVX registers
  - 8 in 32-bit mode
- 256-bit width
  - 8 x FP32
  - 4 x FP64

AVX-512: VADDPS ZMM0, ZMM24, [mem]

- Up to 32 AVX registers
  - 8 in 32-bit mode
- 512-bit width
  - 16 x FP32
  - 8 x FP64

But you need many more features
to use all that real estate effectively…



```
float32 A[N], B[N];

for(i=0; i<8; i++)
{
    A[i] = A[i] + B[i];
}
```

```
float32 A[N], B[N];

for(i=0; i<16; i++)
{
    A[i] = A[i] + B[i];
}
```

AVX-512 Mask Registers

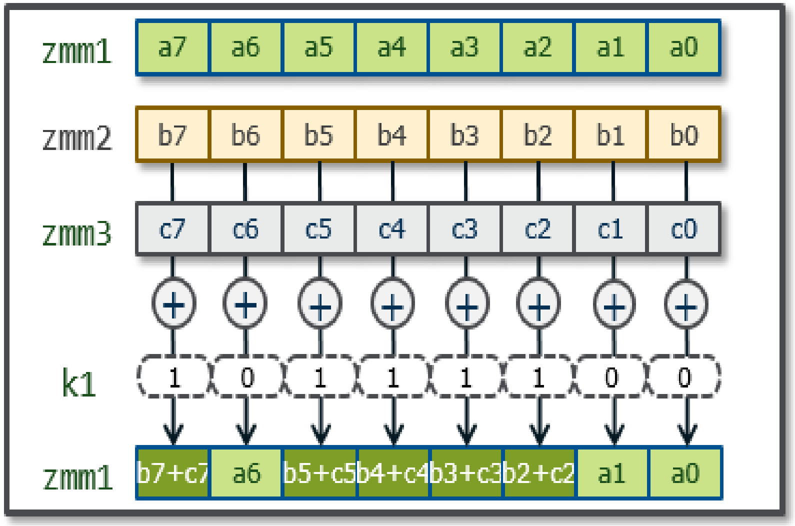
8 Mask registers of size 64-bits

- k1-k7 can be used for predication
  - k0 can be used as a destination or source for mask manipulation operations
  - k0 cannot be used as input mask for vector operations
    - k0 encoding treated as "no mask"

4 different mask granularities.
For instance, at 512b:

- Packed Integer Byte use mask bits [63:0]
  - `VPADDB zmm1 {k1}, zmm2, zmm3`
- Packed Integer Word use mask bits [31:0]
  - `VPADDW zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP32 and Integer Dword use mask bits [15:0]
  - `VADDPS zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP64 and Integer Qword use mask bits [7:0]
  - `VADDPD zmm1 {k1}, zmm2, zmm3`

`VADDPD zmm1 {k1}, zmm2, zmm3`



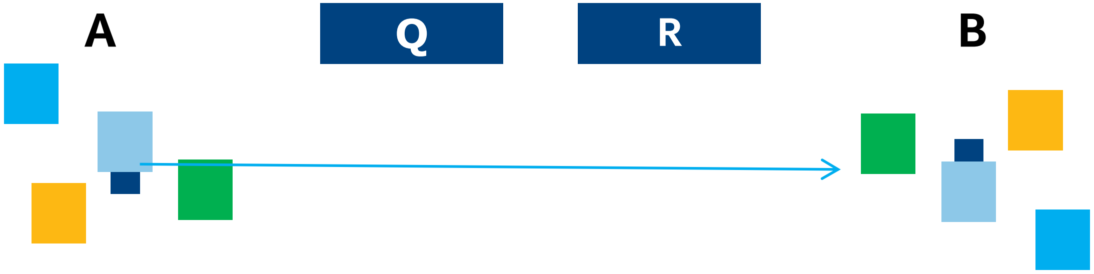
		Vector Length		
		128	256	512
element size	Byte	16	32	64
	Word	8	16	32
	Dw ord/SP	4	8	16
	Qw ord/DP	2	4	8

Gather & Scatter

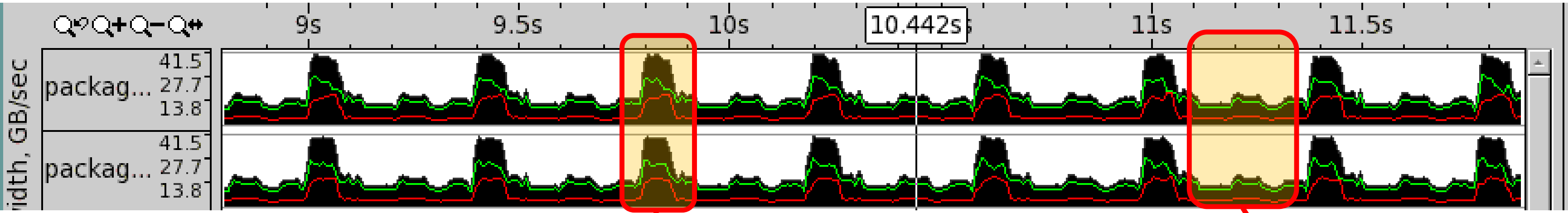
- D/Q/SP/DP element types
- D/Q indices
- Instruction can partially execute
    - k-reg Mask used as completion mask

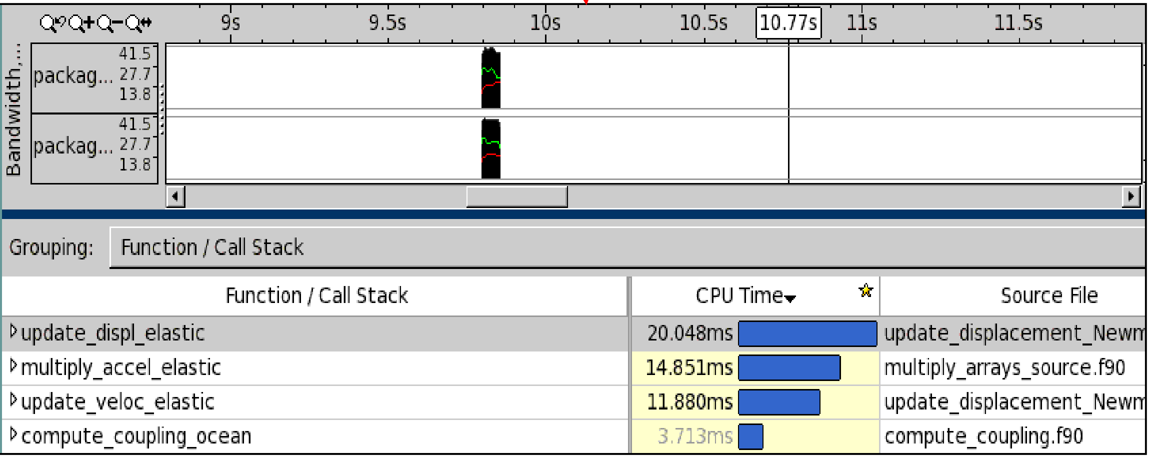
```
for(j=0, i=0; i<N; i++)
{
        B[R[i]] = A[Q[i]];
}
```

```
VMOVDQU64 zmm1, Q[rsi]
VMOVDQU64 zmm2, R[rsi]
VGATHERQQ  zmm0 {k2}, [rax+zmm1*8]
VSCATTERQQ [rax+zmm2*8] {k3}, zmm0
```



Bandwidth Analysis



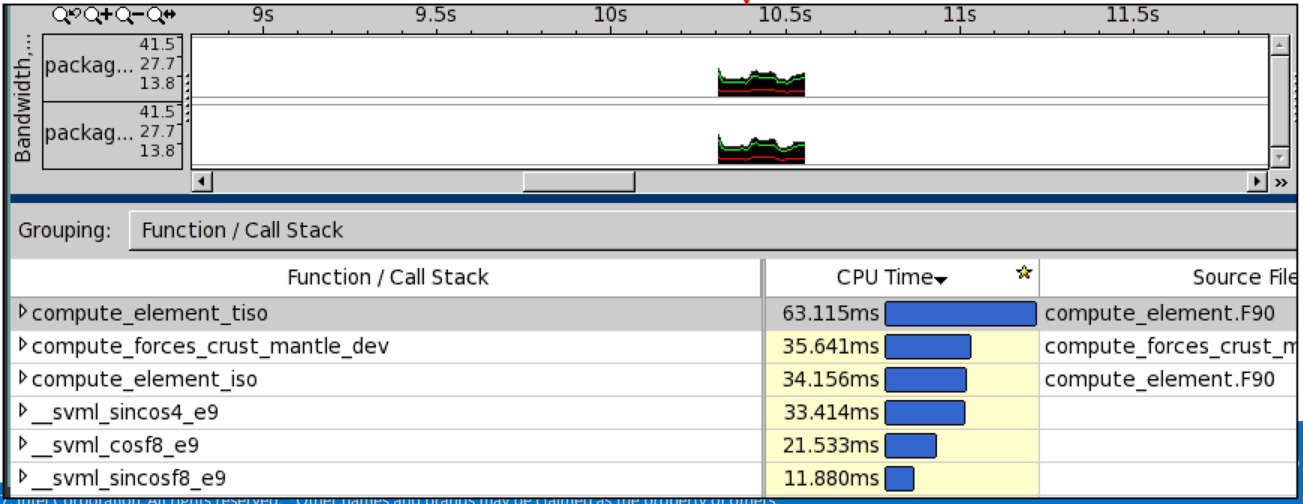


Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Source File
update_displ_elastic	20.048ms	update_displacement_Newm
multiply_accel_elastic	14.851ms	multiply_arrays_source.f90
update_veloc_elastic	11.880ms	update_displacement_Newm
compute_coupling_ocean	3.713ms	compute_coupling.f90

Peak b/w Utilization

Lower b/w Utilization

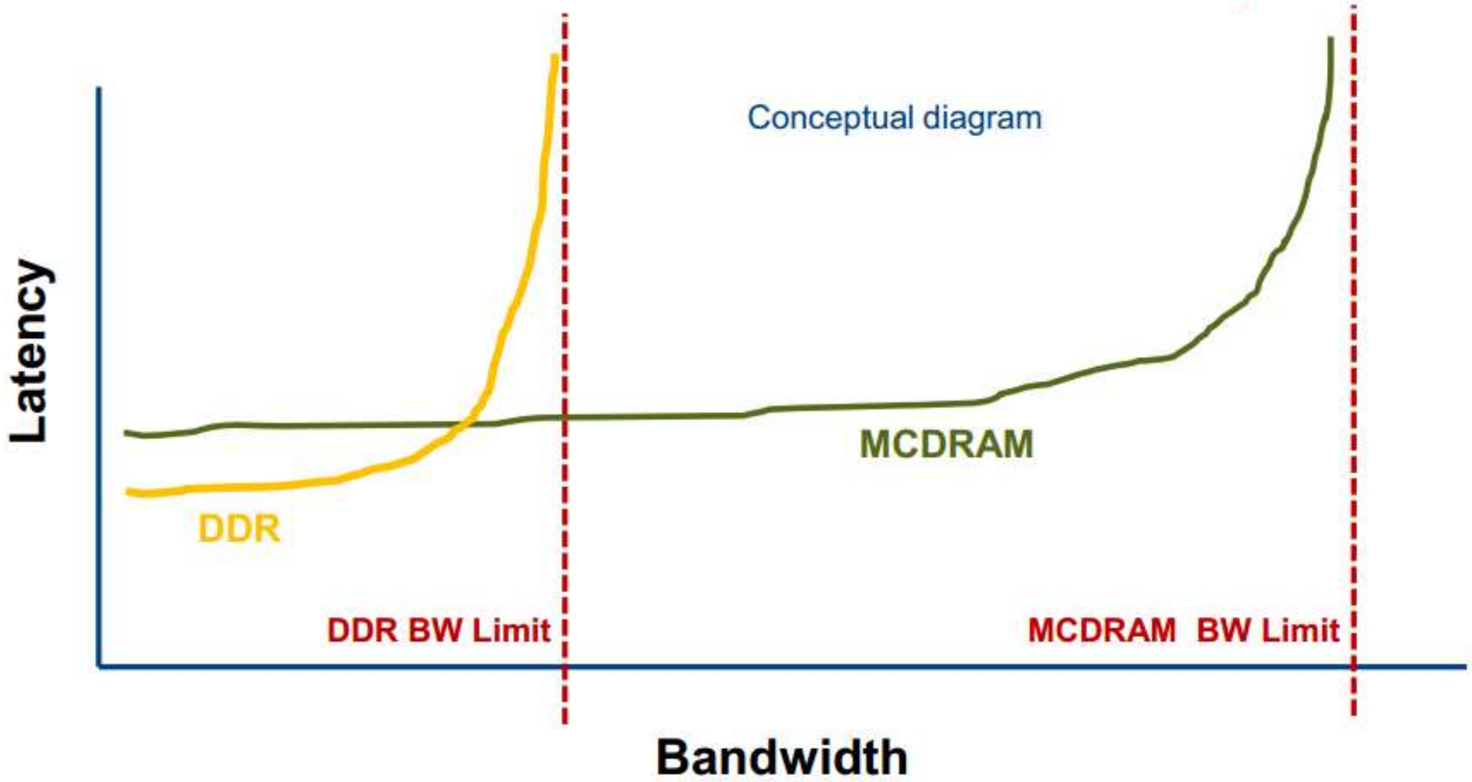


Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Source File
compute_element_tiso	63.115ms	compute_element.F90
compute_forces_crust_mantle_dev	35.641ms	compute_forces_crust_m
compute_element_iso	34.156ms	compute_element.F90
__svml_sincos4_e9	33.414ms	
__svml_cosf8_e9	21.533ms	
__svml_sincosf8_e9	11.880ms	

Not b/w bound for the hotspot loop code!

DDR and MCDRAM Bandwidth vs. Latency



MCDRAM latency more than DDR at low loads but much less at high loads

Optimization Next Steps

- *specfem3D_globe* performance is limited by memory latency issues…try *s/w prefetch or change the data layout*.
- Analyze usage of mixed data-type in a loop… *float* & *double (if any)*.
- Fat loop – causing register pressure, compiler can't vectorize – explore *loop split*…
- Analyze compiler vectorization efficiency.

Code Changes & Gains

Mitigate memory access latency issues: An indirect (random) access was transformed into a unit-stride access. Mesh data in the SPECFEM3D_GLOBE solver is invariant over time/solver steps. Hence, it is a valid transformation to copy data and make it a linear access.

Baseline:

```
xixl = xix(ijk,1,1,ispec)
```

Changed:

```
ia_xix(ijk,1,1,ele_num) = xix(ijk,1,1,ispec)

xixl = ia_xix(ijk,1,1,ele_num)
```

Gain:

~1.40x

2

Code Changes & Gains

Compiler Vectorization – Loop Fission: The compute loops 'iso' and 'tiso' are huge. The compiler is unable to vectorize these loops. So, a manual loop fission was done. A similar effect can be realized by using '!DIR$ DISTRIBUTE POINT' syntax supported by Intel compilers for loop distribution/ fission.

Baseline:

```
 do k=1,NGLLZ
      do j=1,NGLLY
        do i=1,NGLLX
             Loop Body 1
             Loop Body 2
         enddo
...
```

Gain:

~1.69x

Changed:

```
 do k=1,NGLLZ
      do j=1,NGLLY
        do i=1,NGLLX
             Loop Body 1
         enddo
...
 do k=1,NGLLZ
      do j=1,NGLLY
        do i=1,NGLLX
             Loop Body 2
...
```

3

Code Changes & Gains

IVDEP, SIMD directives: Some hotspots in the solver are nested loops with trip counts 5 x 5 and 5 x 25. These are ‘m x m’ loops, matrix-matrix multiplication. The compiler optimization reports (use -qopt-report flag) indicated that not all these loops were vectorized. Using IVDEP or SIMD directives helped the compiler to generate vector code for these loops.

Baseline:

```
 do k=1,NGLLZ
      do j=1,NGLLY
        do i=1,NGLLX
           do l=1,5
              Loop Body
         enddo
...
```

Changed:

```
 do k=1,NGLLZ
!$OMP SIMD PRIVATE(j,l)
     do i=1,NGLLX
        do j=1,NGLLY
           do l=1,5
              Loop Body
         enddo
...
```

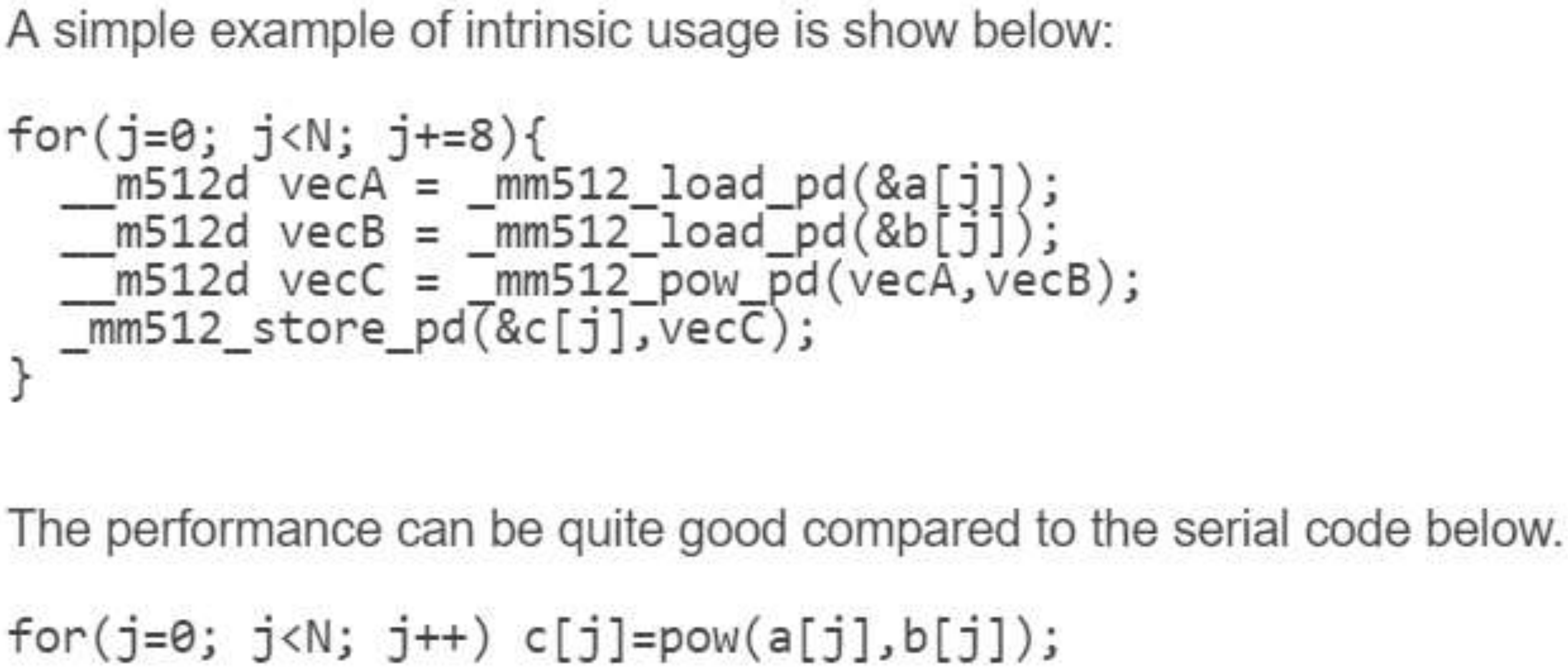
Gain:

~2.09x

4

Code Changes & Gains

Writing “low-level” Intrinsic functions



A simple example of intrinsic usage is show below:

```
for(j=0; j<N; j+=8){
   __m512d vecA = _mm512_load_pd(&a[j]);
   __m512d vecB = _mm512_load_pd(&b[j]);
   __m512d vecC = _mm512_pow_pd(vecA,vecB);
   _mm512_store_pd(&c[j],vecC);
}
```

The performance can be quite good compared to the serial code below.

```
for(j=0; j<N; j++) c[j]=pow(a[j],b[j]);
```

http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017

Optimization Summary

Use compiler optimization options...

Do analysis of your code to find hotspots.

Optimize by code changes which include:

- ✓ Data transformation for mitigating access latencies
- ✓ Compiler vectorization and loop optimizations
- ✓ Data alignment and padding for arrays
- ✓ Redundant compute elimination
- ✓ IVDEP, SIMD directives usage

and more...

AMBER WRF VISIT VASP UTBENCH SU2 SG++ SeisSol ROTOR SIM R Quantum Espresso Optimized integral OPENMP/ MPI Openflow NWChem

Modernizing HPC Community Codes

AVBP (Large Eddy)

Blast

BUDE

CAM-5

CASTEP

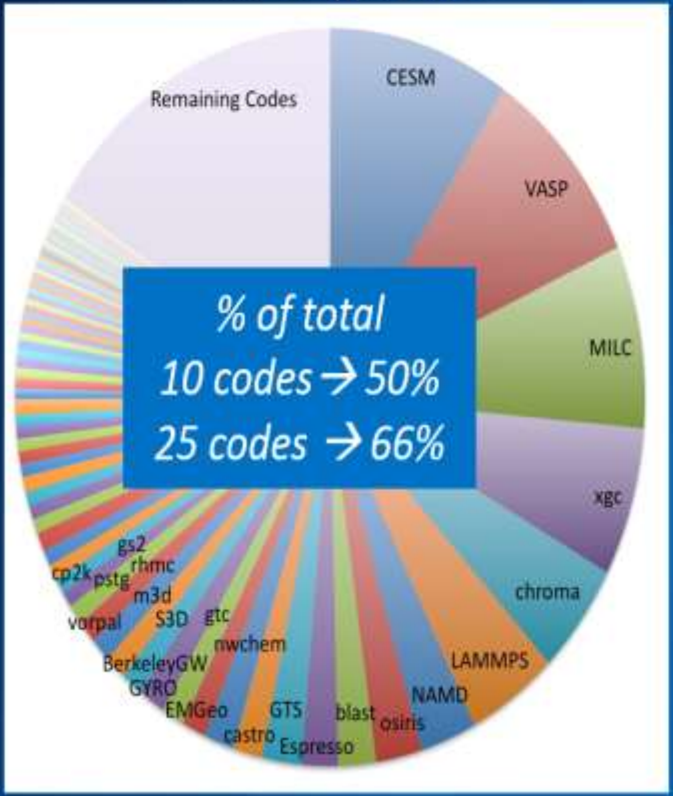
CESM

CFSv2

CIRCAC

Breakdown of Application Hours

NERSC - Hopper 2012[1]



% of total

10 codes → 50%

25 codes → 66%

Intel® Parallel Computing Centers

Collaborating to accelerate the pace of discovery

>40 Centers

13 Countries

>70 Codes

2 User Groups

https://software.intel.com/en-us/ipcc

NEMO5

MPAS

Mardyn

MACPO

Ls1

Harmonie

GTC

GS2

Gromacs

GPAW

CliPhi (COSMOS) COSA Cosmos codes DL-MESO DL-Poly ECHAM6 Elmer FrontFlow/Blue Code GADGET GAMESS-US

[1] Source: NERSC

Thank you!