Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor

Douglas Doerfler, Jack Deslippe, Samuel Williams, Leonid Oliker, Brandon Cook, Thorsten Kurth, Mathieu Lobet, Tareq Malas, Jean-Luc Vay, and Henri Vincenti

Lawrence Berkeley National Laboratory

ISC-2016 IXPUG Workshop
June 23rd, 2016
Frankfurt, Germany

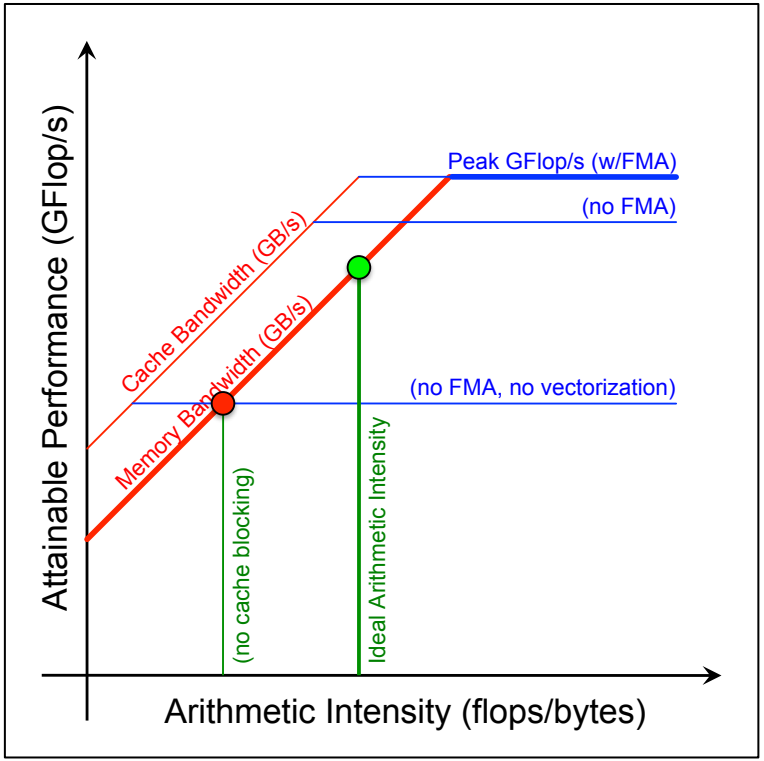
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB | NERSC

Introduction

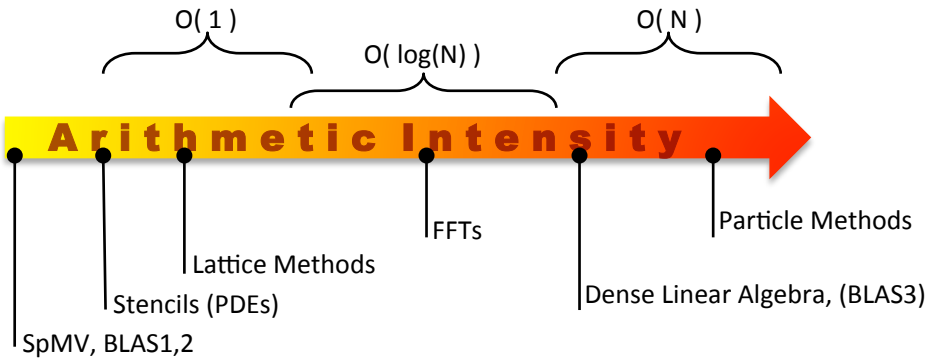
- The challenge of porting to any new architecture is gaining an understanding of the architectural bottlenecks that may be exposed to an application code
- This is especially true for a new many-core processor architecture such as Intel's Knights Landing, as it provides multiple unique features such as (up to) 288 HW threads, dual 512-bit vector units and on-package, high-bandwidth memory
- The Roofline Performance Model provides an important framework for the optimization process with code teams

The Roofline Model



$$\text{Attainable FLOPs / sec} = \min \begin{cases} \text{Peak FLOPs / sec}, \\ \text{Peak Memory Bandwidth} \; x \; \text{Arithmetic Intensity} \end{cases}$$

$$\text{Arithmetic Intensity} = \frac{\text{Total FLOPs}}{\text{Total Bytes}}$$



Target Hardware Architecture

- Intel KNL
  - Standalone Intel white boxes
  - KNL preproduction, B0 stepping
  - 64 cores @ 1.3 GHz, 4 hyper-threads/core
  - 16 GB MCDRAM (>460 GB/s peak BW)
  - 96 (6x16) GB DDR4 @ 2133 GHz (102 GB/s peak BW)
- Intel Haswell (baseline)
  - Cori Phase 1 supercomputer at NERSC
  - Single node tests
  - Dual socket, 16 cores/socket @ 2.3 GHz
  - 128 (8x16) GB DDR4 @ 2133 GHz (137 GB/s peak BW)

Berkeley Empirical Roofline Toolkit Method

Method:

- Sweep a range of FLOPs per iteration
  - 1 FLOP/iteration
    - Sweep a range of MPI ranks, keeping total number of threads constant
    - 1 rank, 64 threads
    - 2 ranks, 32 threads
    - Etc.
  - 2 FLOPs/iteration
    - Same as 1 FLOP
  - ... up to e.g. 64 FLOPs/iteration
- Sweep over a range of array sizes
  - E.g. 1 KB to 1 GB
- Choose the largest values for L1, L2, DRAM and GFLOP/s out of the entire sweep

Toolkit Kernel1:

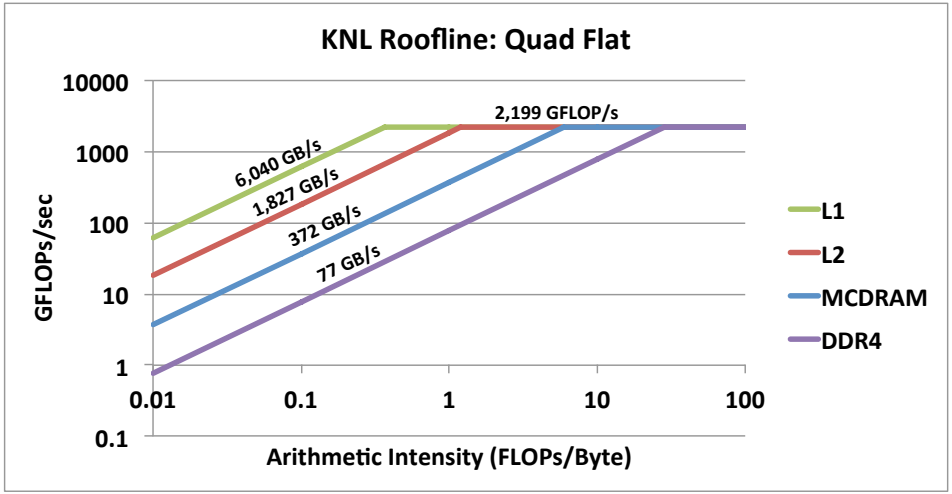
Loop for 1 FLOPs/iteration:

```
for (i=0; i<array_length; ++i) {
  beta = A[i] + alpha;
  A[i] = beta;
}
```

Loop for 2 or more FLOPs/iteration:
(Note: the higher level loop is unrolled to allow the compiler to vectorize despite the vector dependency)

```
for (i=0; i<array_length; ++i) {
  beta = 0.8;
  beta = beta * A[i] + alpha;
  <repeat to achieve N FLOPs>
  A[i] = beta;
}
```

KNL Roofline Results



- Using 2 threads/core
- Max L1, L2 and MCDRAM
  - 1 FLOP/iteration
  - 4 MPI + 32 threads
- Max GFLOP/s
  - 64 FLOPs/iteration
  - 2 MPI + 64 threads

	Quad Cache	Quad Flat	SNC2	SNC4	Peak[a]
GFLOP/s	2,205	2,199	2,224	2,212	2,253
L1	5,894	6,040	5,889	6,055	9,011
L2	1,834	1,827	1,829	1,840	2,252[b]
MCDRAM	345	372	381	415	420[c]
DDR		77.0	76.9	76.9	102

All Bandwidths are in GB/s

(a) Values assume an AVX frequency of 1.1 GHz

(b) L2 assumed ~(L1 / 4)?

(c) MCDRAM BW is for 1R/1W per iteration

Applications, Proxies and Kernels

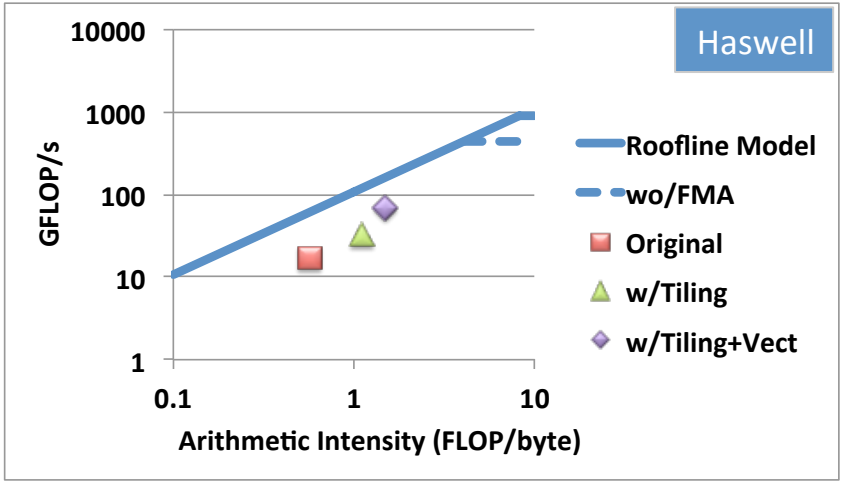
- PICSAR (proxy for WARP)
  - Particle-in-cell (PIC) code designed to simulate charged particle beams and laser-matter interaction
- EMGeo (SpMV kernel)
  - Geophysical imaging, studying medium properties by performing scattering experiments using electromagnetic or seismic waves
- MFDn (SpMV kernel)
  - Many-body Fermion Dynamics for nuclei, nuclear physics
- BerkeleyGW
  - Materials science, computes excited state properties of materials

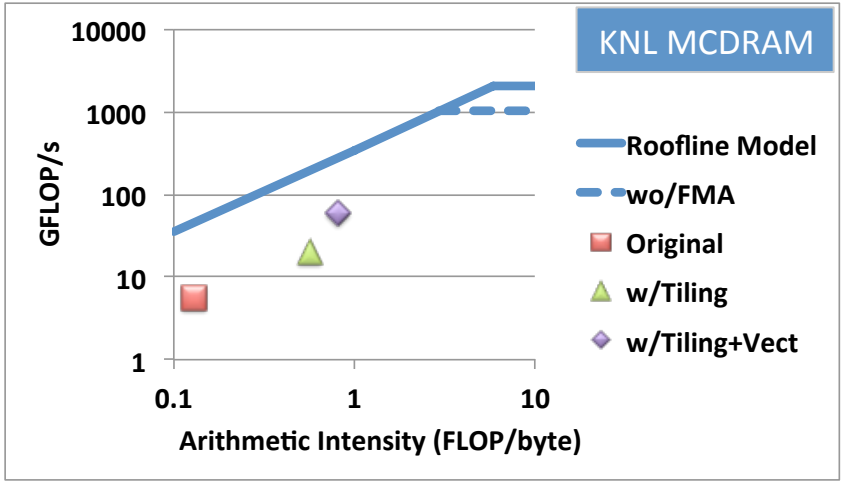
PICSAR Optimizations

- A Fortran kernel based on WARP, a test bed for profiling and optimizations
- The current deposition and field gathering steps is the focus of this analysis
- Optimizations
  - Original code spatially decomposes the problem with MPI
  - MPI subdomains are subdivided into tiles handled with OpenMP, improving memory locality and hence cache reuse of tiles. Having a large number of tiles allows OpenMP threads to load balance across tiles.
  - Direct current deposition and field gathering interpolation steps were rewritten to enable more efficient vectorization, plus particle cell sorting was added to again improve memory locality and hence cache reuse
- Test Case
  - Maxwellian homogeneous plasma with initial thermal velocity of 0.1c.
  - Domain discretization is 100x100x100 cells with 20 super particles per cell
  - Haswell: 2 MPI ranks + 16 OpenMP threads/rank
  - KNL: 4 MPI ranks + 32 OpenMP threads/rank (2 threads/core)

PICSAR Performance

	Haswell		KNL MCDRAM		KNL DDR		
Optimization	AI	GFLOP/s	AI	GFLOP/s	AI	GFLOP/s	KNL Speedup
Original	0.57	16.7	0.13	5.6	0.13	5.4	0.34
+Tiling	1.10	32.0	0.56	20.0	0.56	19.2	0.63
+Tiling +Vect	1.50	67.5	0.81	60.4	0.81	49.4	0.89





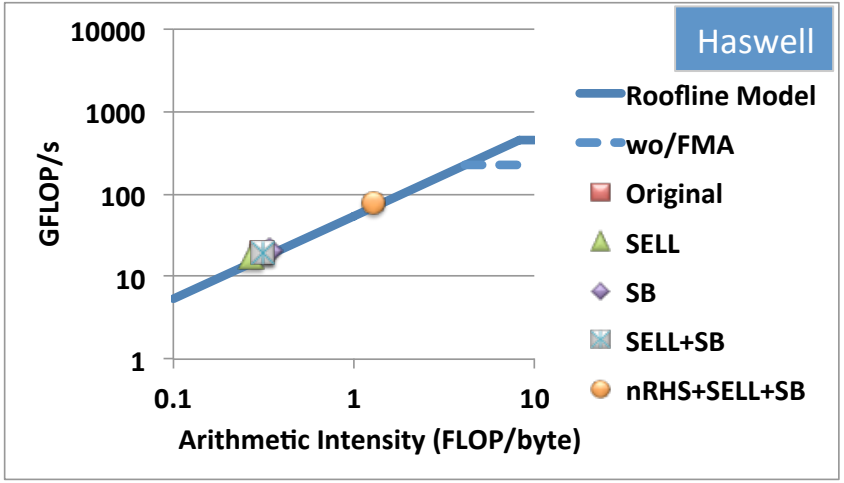
EMGeo Optimizations

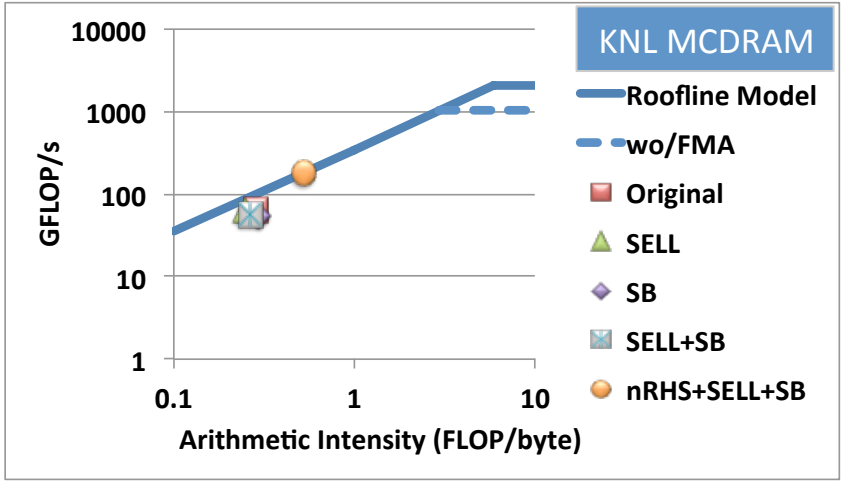
- Focus on the seismic part and forward step of the inverse scattering problem
- Sparse matrix-vector (SpMV) kernel dominates total runtime, ~70% for Haswell and 50% for KNL.
- Three optimizations
  - Sliced ELLPack (SELL) sparse matrix format provides a more efficient matrix format, reduces the FLOP count and reduces data transferred from DRAM
  - Spatial Blocking (SB) to increase memory locality
  - Multiple right hand sides (nRHS) cache blocking to increase memory locality
- Test Case
  - Benchmark focuses on a matrix with a maximum 12 non-zeros per row
  - Production code evaluates about 256 independent RHS, we evaluate 32 RHS in Haswell and 64 in KNL. However, the results are normalized for comparison
  - Kernel has no MPI, hence Haswell result uses a single socket to avoid NUMA issues

EMGeo Performance

	Haswell[1]		KNL MCDRAM		KNL DDR		
Optimization	AI	GFLOP/s	AI	GFLOP/s	AI	GFLOP/s	KNL Speedup
Original	0.31	19.2	0.27	71.1	0.27	23.5	3.7
+SELL only	0.27	16.9	0.24	71.0	0.24	21.2	4.2
+SB only	0.34	20.2	0.28	62.3	0.28	20.9	3.1
+SELL+SB	0.31	19.2	0.26	63.9	0.26	19.6	3.3
+nRHS+SELL+SB	1.29	77.7	0.76	278.5	0.76	65.8	3.6

1) EMGeo kernel used for this analysis is not an MPI code, a single Haswell socket was used to avoid NUMA issues





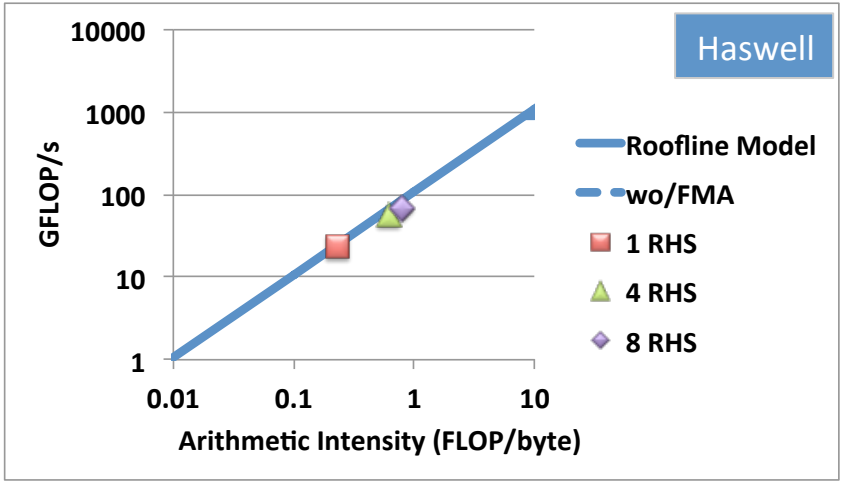
MFDn Optimizations

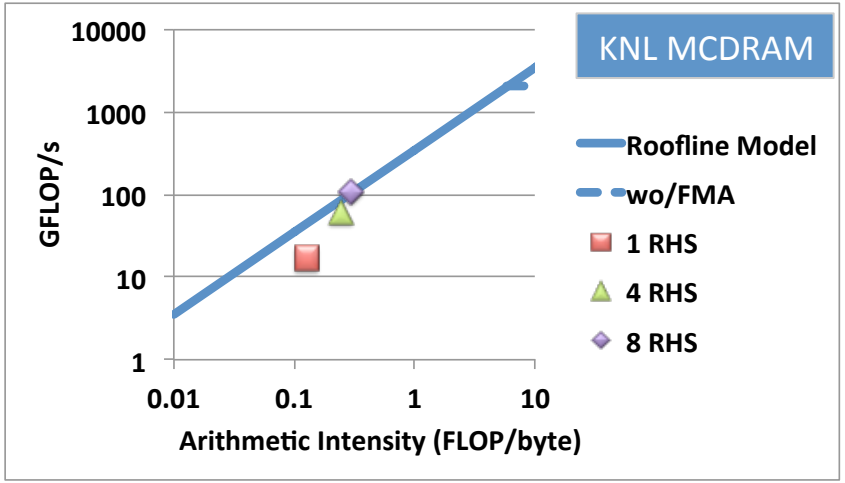
- Sparse matrix is stored in compressed sparse block coordinate (CSB_COO) format
- Dominated by SpMV and SpM^TV
- Optimizations
  - Replace SpMV with SpMM on blocks of vectors to improve vectorization
  - Vectors occupy MCDRAM and blocked (nRHS) to improve bandwidth and locality (the larger sparse matrix resides in DDR4)
- Test case
  - 2 protons and 6 neutrons
  - Target test problem of size *nxn* with *n*=1e10 and a local sparsity of 5e-7, resulting in ~7.5e9 nonzero elements

MFDn Performance[1]

	Haswell		KNL MCDRAM[2]		KNL DDR		
nRHS	AI	GFLOP/s	AI	GFLOP/s	AI	GFLOP/s	KNL Speedup
1	0.23	23.2	0.13	17.1	0.13	13.5	0.74
4	0.62	56.8	0.25	62.4	0.25	27.8	1.1
8	0.80	67.5	0.30	109.1	0.30	30.7	1.6

1) MFDn uses single precision only, Roofline model is adjusted accordingly
2) Stores input/output vectors in MCDRAM, all other data resides in DDR



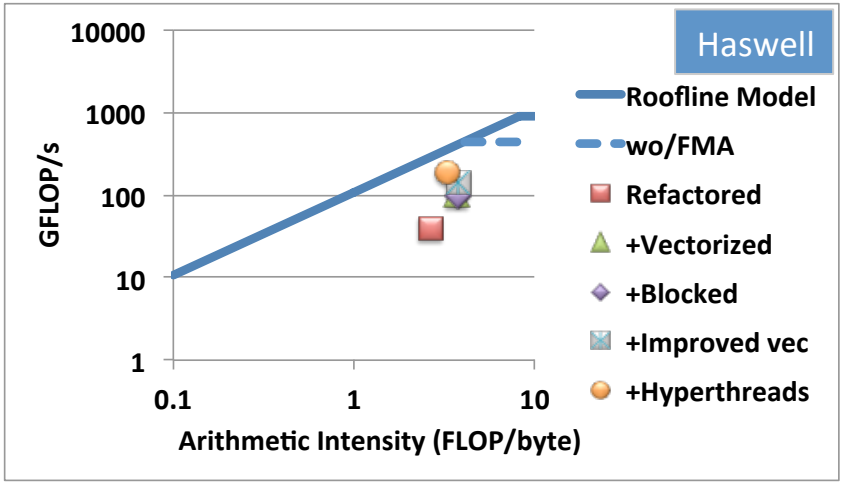


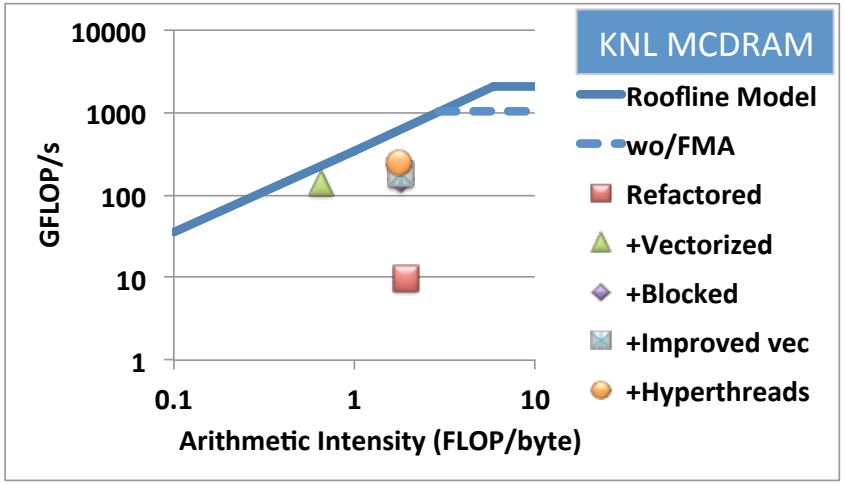
BerkeleyGW Optimizations

- Dense linear algebra (GEMM, diagonalization and inversion) dominate
- Focus on hand tuned code for this analysis (FFTs and linear algebra steps excluded)
- Optimizations
  - Refactor to support OpenMP and improved locality
    - Outer loop decomposed over MPI
    - Nested inner loops decomposed over OpenMP w/vectorization
  - Support for compiler auto-vectorization by reordering the nested loops
  - Add cache blocking to potentially better utilize LLC
  - Replace the complex divide with a real absolute value to avoid x87 instructions
  - Add hyper-threading, and replace above complex divide optimization with *–fp-model-fast=2*
- Test Case
  - ?

BerkeleyGW Performance

	Haswell		KNL MCDRAM		KNL DDR		
Optimization	AI	GFLOP/s	AI	GFLOP/s	AI	GFLOP/s	KNL Speedup
Refactored	2.64	38.7	1.93	9.80	1.93	9.80	0.25
+ Vectorized	3.68	100.3	0.66	143.4	0.66	55.1	1.43
+ Blocked	3.77	100.3	1.79	153.2	1.79	140.8	1.53
+ Improved Vect	3.78	142.6	1.80	178.4	1.80	142.1	1.25
+ Hyper-threads	3.27	186.9	1.76	252.6	1.76	144.0	1.35





Performance Summary

	GFLOP/s			Speedup	
	Haswell (HSW)	KNL MCDRAM	KNL DDR	KNL/HSW	MCDRAM/DDR
PICSAR	67.5	60.4	49.4	0.89	1.2
EMGeo (SpMV)	77.7	181.0	43.6	2.33	4.2
MFDn	67.5	109.1	30.7	1.62	3.6
BerkeleyGW	186.9	252.6	144.0	1.35	1.75

- All codes showed a significant performance gain vs. the baseline version
- KNL outperforms Haswell with the exception of PICSAR
  - however the PICSAR optimizations benefited both architectures significantly
- No code showed to be peak floating-point bound, all were in the regime where the Roofline ceiling was bandwidth
  - EMGeo and MFDn were clearly bandwidth bound
  - PICSAR and BerkeleyGW showed potential for further optimization
- Haswell consistently attains a higher arithmetic intensity than KNL.
  - KNL generally moves more data to/from memory than Haswell
  - The higher theoretical performance benefits of MCDRAM bandwidth may not be fully realized due to this extra data movement, and exploration of the performance benefits of a larger last-level caches are in order

Thank You

dwdoerf@lbl.gov