DEEP DIVE MEETING INTO XGC

Argonne NATIONAL LABORATORY

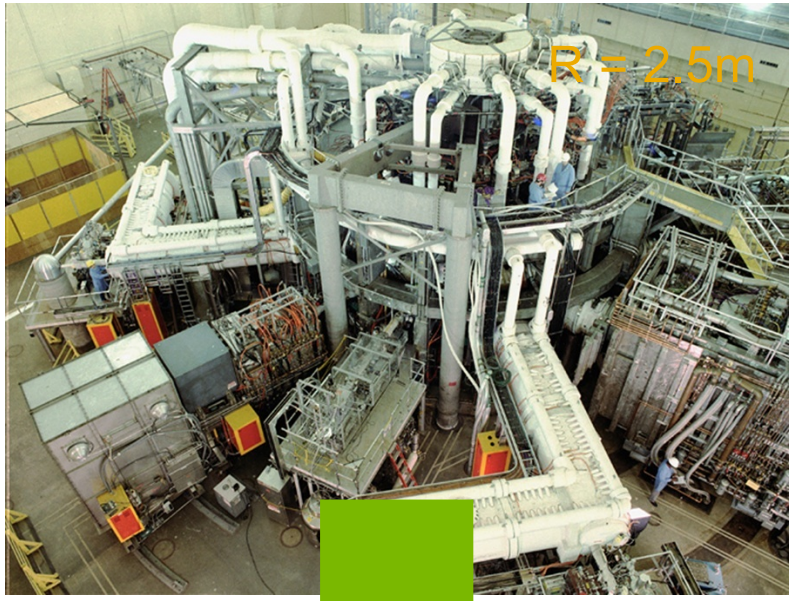
WHAT CAN KNL DO FOR YOU?

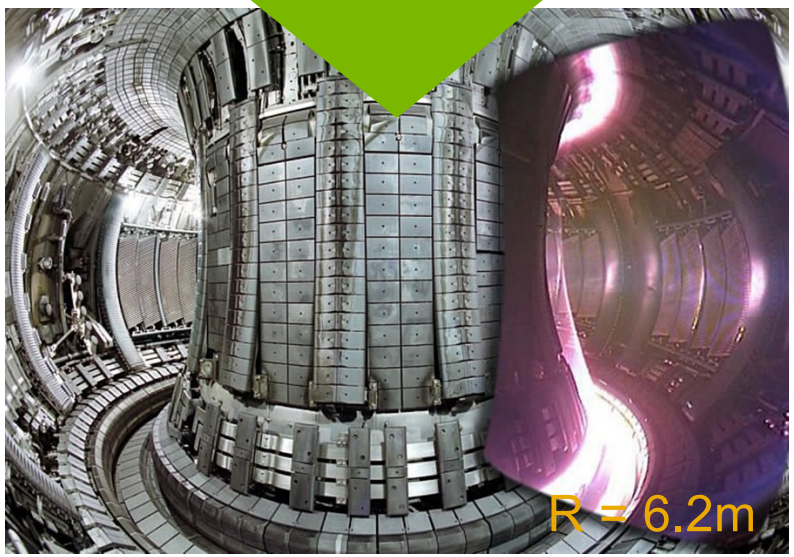
BRIAN MACKIE-MASON
Aurora ESP Postdoc
Argonne Leadership Computing Facility

Decmeber 11-12, 2018
Princeton Plasma Physics Laboratory, NJ

MOTIVATION

Science & Computation





1 eFLOPs

Argonne NATIONAL LABORATORY

OUTLINE

Topics covered

- Architectural Overview
- Overview of performance opportunities
- Performance optimization results
- Aurora
- Conclusions

INTEL KNIGHTS LANDING ARCHITECTURE

MOTIVATION

Continuation of Moore's Law

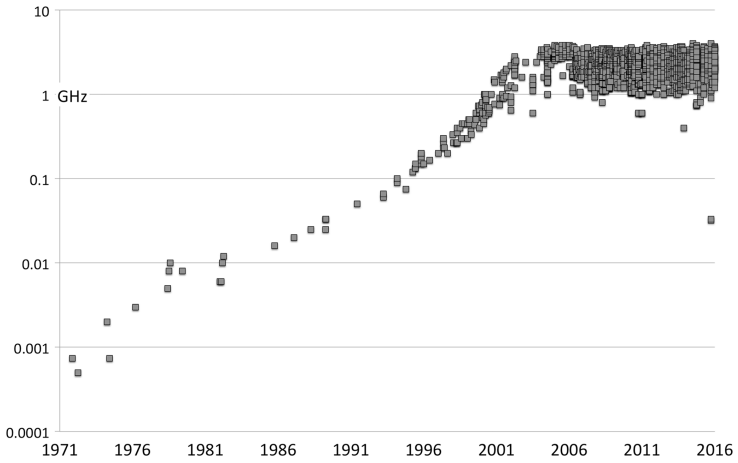


Figure 1: Processor speed

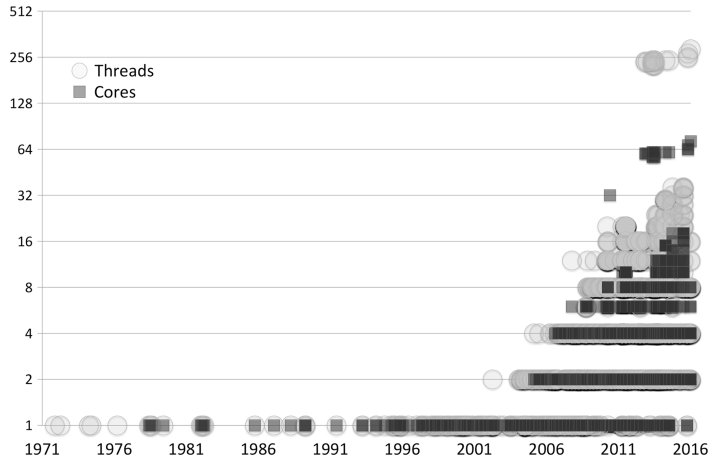


Figure 2: Multi-threading

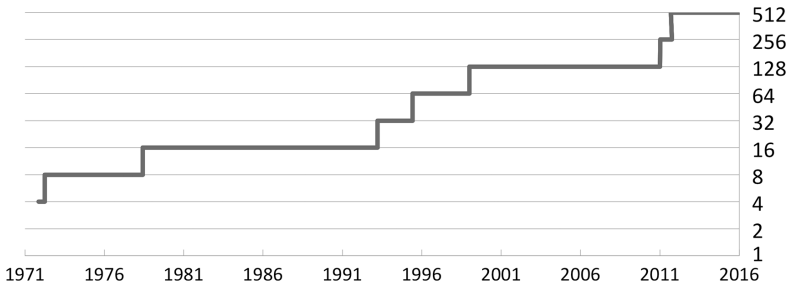


Figure 3: Vector Length

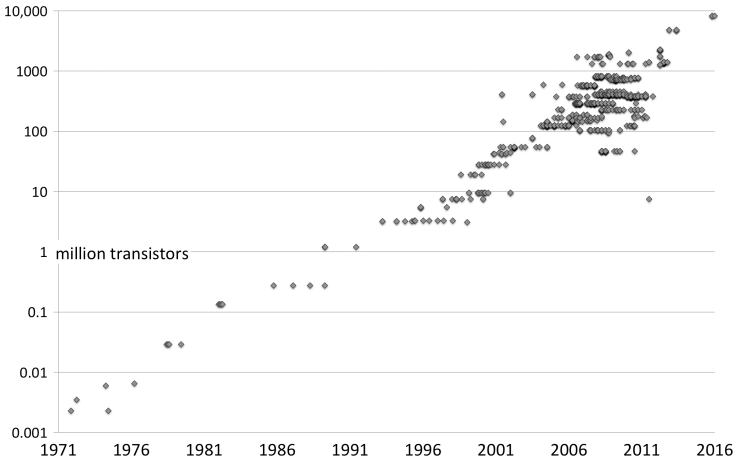
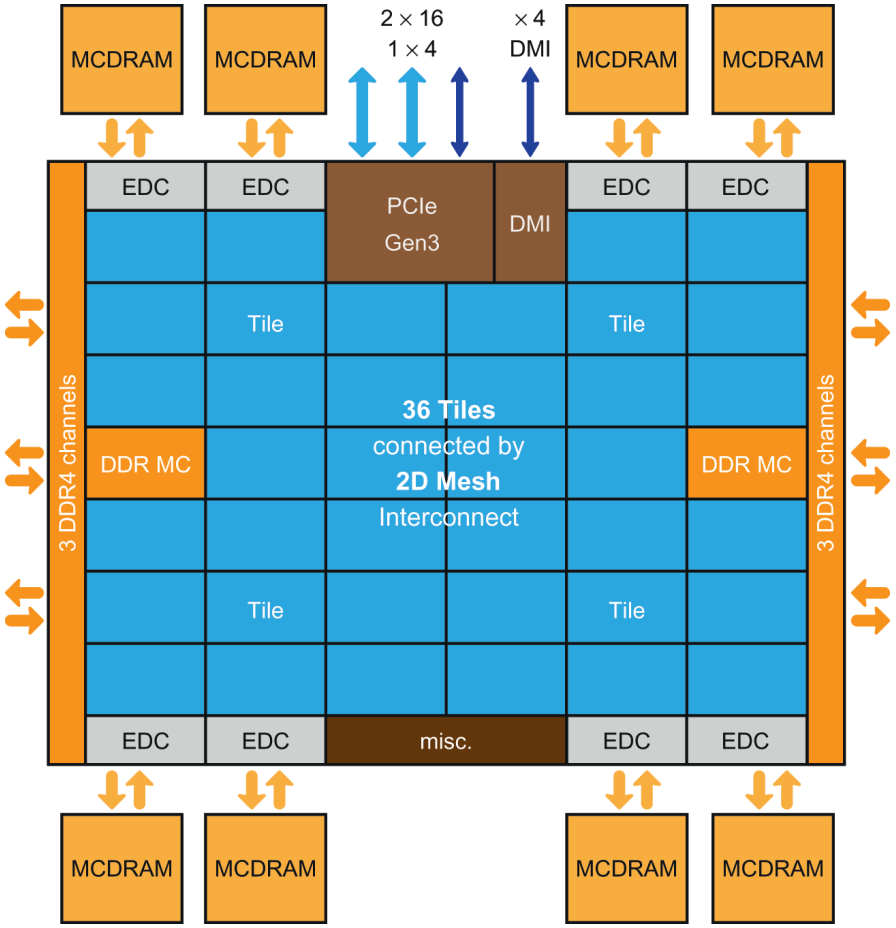
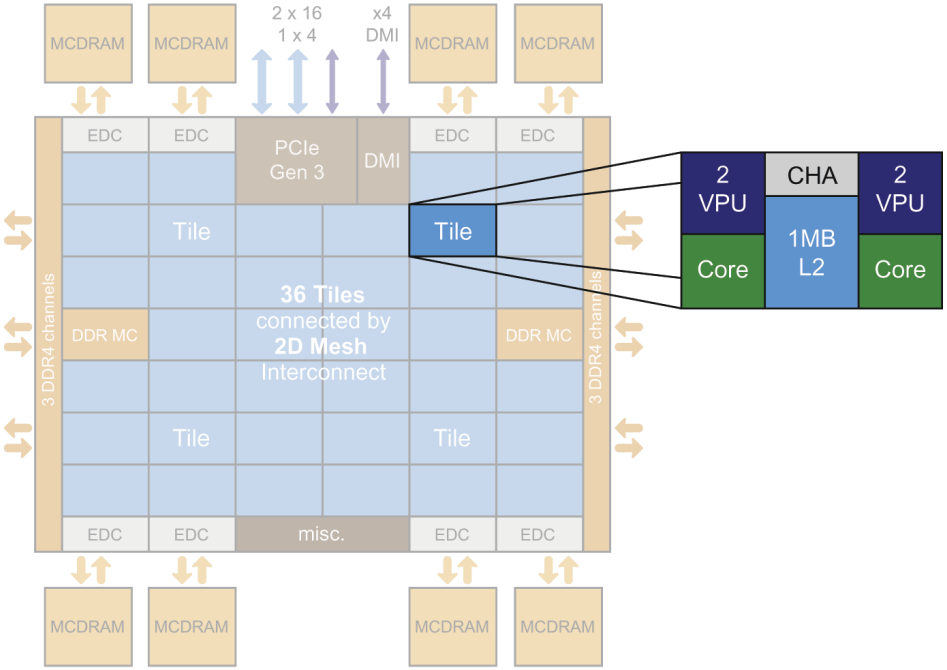


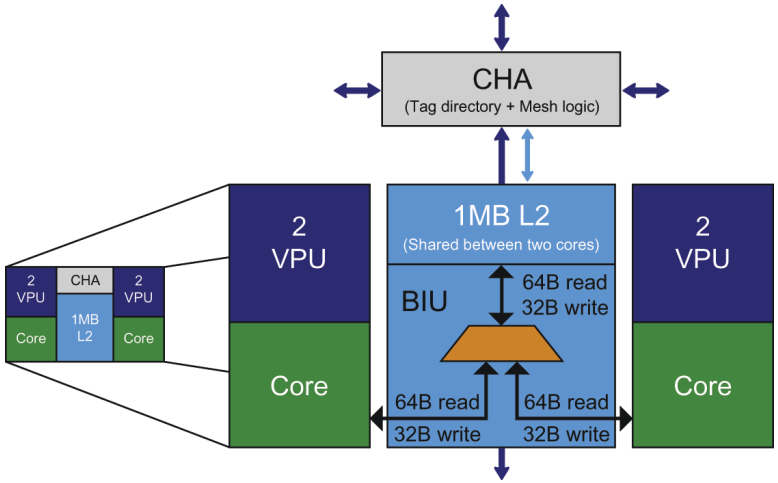
Figure 4: Moore's Law Revised

KNL ARCHITECTURE

Overview







Argonne NATIONAL LABORATORY

SYSTEMS OVERVIEW

Top500 List

Mode: TOP500

List: November 2018 | Country: All | Vendor: All | Processor Generation: Intel Xeon Phi | Architecture: All | Segment: All | Application Area: All | Interconnect Family: All | OS Family: All | Rank from: 1 | Rank to: 500

Submit

8 entries found.

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
6	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/NNSA/LANL/SNL United States	979,072	20,158.7	41,461.2	7,578.1
12	**Cori** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. DOE/SC/LBNL/NERSC United States	622,336	14,014.7	27,880.7	3,939
13	**Nurion** - Cray CS500, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Cray Inc. Korea Institute of Science and Technology Information Korea, South	570,020	13,929.3	25,705.9	
14	**Oakforest-PACS** - PRIMERGY CX1640 M1, Intel Xeon Phi 7250 68C 1.4GHz, Intel Omni-Path , Fujitsu Joint Center for Advanced High Performance Computing Japan	556,104	13,554.6	24,913.5	2,718.7
16	**Tera-1000-2** - Bull Sequana X1000, Intel Xeon Phi 7250 68C 1.4GHz, Bull BXI 1.2 , Bull, Atos Group Commissariat a l'Energie Atomique (CEA) France	561,408	11,965.5	23,396.4	3,178
17	**Stampede2** - PowerEdge C6320P/C6420, Intel Xeon Phi 7250 68C 1.4GHz/Platinum 8160, Intel Omni-Path , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	367,024	10,680.7	18,309.2	
19	**Marconi Intel Xeon Phi** - CINECA Cluster, Lenovo SD530/S720AP, Intel Xeon Phi 7250 68C 1.4GHz/Platinum 8160, Intel Omni-Path , Lenovo CINECA Italy	348,000	10,384.9	18,816.0	
24	**Theta** - Cray XC40, Intel Xeon Phi 7230 64C 1.3GHz, Aries interconnect , Cray Inc. DOE/SC/Argonne National Laboratory United States	280,320	6,920.9	11,661.3	
48	**ONYX** - Cray XC40, Intel Xeon Phi 7230 64C 1.3GHz/Xeon E5-2699v4, Aries interconnect , Cray Inc. ERDC DSRC United States	160,304	3,409.6	5,865.5	
60	**Camphor 2** - Cray XC40, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc. Academic Center for Computing and Media Studies (ACCMS), Kyoto University Japan	122,400	3,057.3	5,483.5	748.1
107	**scafellpike** - Bull Sequana X1000, Intel Xeon Phi 7210 64C 1.3GHz/Xeon Gold 6142, Mellanox EDR , Bull, Atos Group Science and Technology Facilities Council United Kingdom	80,640	1,829.9	4,329.7	529
211	**Endeavor** - Intel Cluster, Intel Xeon Gold 6148/Xeon Phi 7250F 68C 1.4GHz, Intel Omni-Path , Intel Intel United States	45,680	1,463.0	2,507.3	431
242	**JOLIOT-CURIE KNL** - Bull Sequana X1000, Intel Xeon Phi 7250 68C 1.4GHz, Bull BXI 1.2 , Bull, Atos Group CEA/TGCC-GENCI France	56,304	1,311.3	2,339.6	326
370	Apollo 6000 XL260a, Intel Xeon Phi 7230 64C 1.3GHz, Infiniband EDR , HPE Energy Company (D) United States	41,600	1,071.4	1,730.6	
371	Apollo 6000 XL260a, Intel Xeon Phi 7230 64C 1.3GHz, Infiniband EDR , HPE Energy Company (D) United States	41,600	1,071.4	1,730.6	
372	**BeBop** - Cray CS400, Intel Xeon Phi 7230 64C 1.3GHz/Xeon E5-2695v4, Intel Omni-Path , Cray Inc. DOE/SC/Argonne National Laboratory United States	46,720	1,070.6	1,750.0	
394	**TX-Green** - S7200AP Cluster, Intel Xeon Phi 7210 64C 1.3GHz, Intel Omni-Path , Dell EMC MIT/Lincoln Laboratory United States	41,472	1,032.8	1,725.2	
454	**Sequana BXI** - Bull Sequana X1000, Intel Xeon Phi 7250 68C 1.4GHz, Bull BXI 1.2 , Bull, Atos Group Atos France	35,360	947.9	1,584.1	212

SYSTEMS OVERVIEW

XGC Allocations

- Cori-KNL

KNL Compute nodes	9,688	Each node is a single-socket Intel® Xeon Phi™ Processor 7250 ("Knights Landing") processor with 68 cores per node @ 1.4 GHz
		Each core has two 512-bit-wide vector processing units. Each core has 4 hardware threads (272 threads total). Two cores form a tile.
		44.8 GFlops/core; 3 TFlops/node; 29.5 PFlops total (theoretical peak)
		Each node has 96 GB DDR4 2400 MHz memory, six 16 GB DIMMs (102 GiB/s peak bandwidth). Total aggregate memory (combined with MCDRAM) is 1.09 PB.
		Each node has 16 GB MCDRAM (multi-channel DRAM), > 460 GB/s peak bandwidth
		Each core has its own L1 caches, with 64 KB (32 KiB instruction cache, 32 KB data). Each tile (2 cores) shares a 1MB L2 cache.
Interconnect		Cray Aries with Dragonfly topology with 5.625 TB/s global bandwidth (Phase I). 45.0 TB/s global peak bisection bandwidth (Phase II).

- Theta-KNL

The system is equipped with 4,392 nodes, each containing a 64 core processor with 16 gigabytes (GB) of high-bandwidth in-package memory (MCDRAM), 192 GB of DDR4 RAM, and a 128 GB SSD. Theta's initial parallel file system is 10 petabytes.

Theta has several features that allow scientific codes to achieve higher performance, including:

- High-bandwidth MCDRAM (300 - 450 GB/s depending on memory and cluster mode), with many applications running entirely in MCDRAM or using it effectively with DDR4 RAM
- Improved single thread performance
- Potentially much better vectorization with AVX-512
- Large total memory per node

Theta System Configuration

- 24 racks
- 4,392 nodes
- 281,088 cores
- 70.272 TB MCDRAM
- 843.264 TB DDR4
- 562.176 TB SSD
- Aries interconnect with Dragonfly configuration
- 10 PB Lustre file system
- Peak performance of 11.69 petaflops

Argonne NATIONAL LABORATORY

SYSTEM/ARCHITECTURE OVERVIEW

Key takeaways

- Many-integrated core architecture (MIC) key development
- Maintains the rapid progression of compute power for computational science
- Raw compute speed no longer only factor
  - Computational scientists should use advanced architectures
  - Understanding of how to make use of architecture useful
- Hierarchical memory arrangement
  - Large and slow vs. small and fast
  - Programming methodology should harness this
- Efficient power consumption

KNL OPTIMIZATION OPPORTUNITIES

THE BEST WAY TO WRITE EFFICIENT MODERN CODE

Three competing theories

1. Write code in 2005 that is optimized for systems in 20XY
2. Re-write code from 2005 in 20XY based on current systems
3. Make small- to medium-sized re-factoring changes to code written in 2005 that optimize performance in 20XY

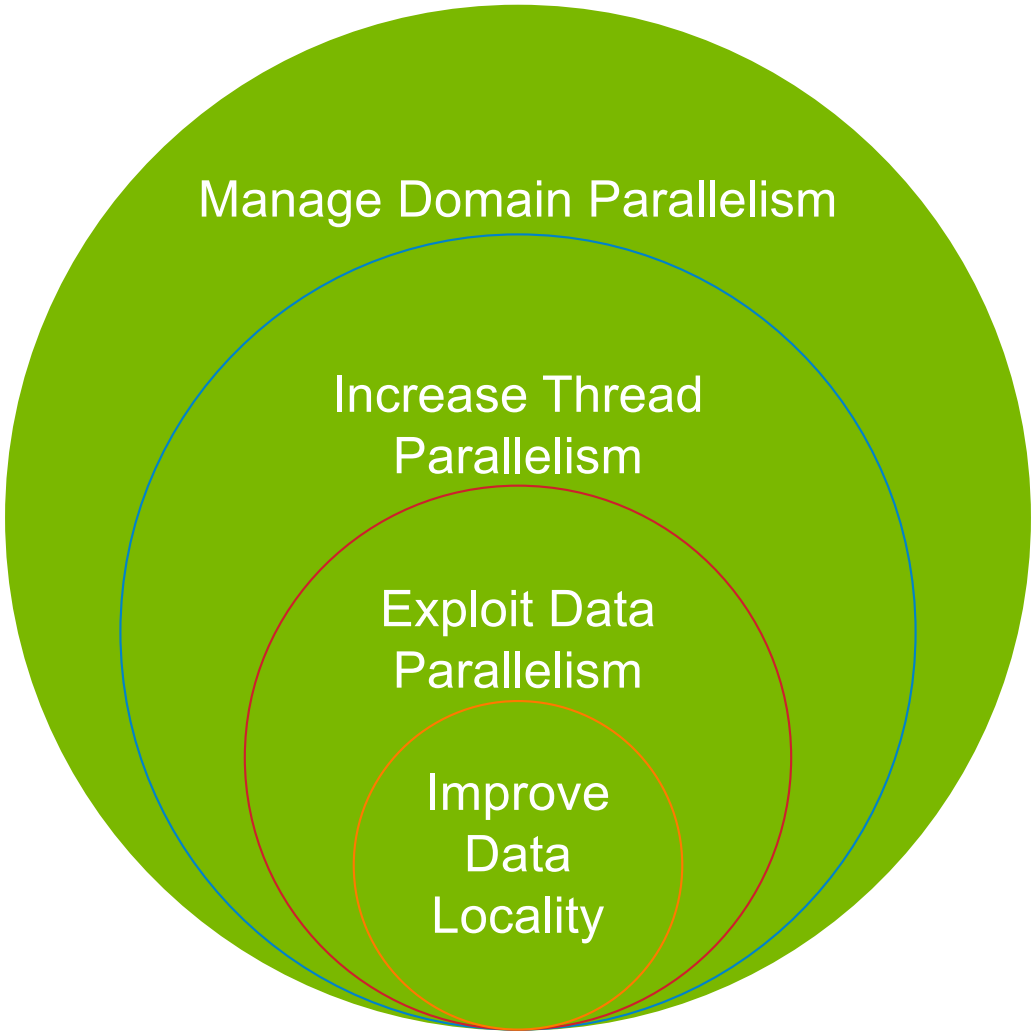
OPTIMIZATION PROCEDURE

General Advice

1. Workflow
   a. Turn off vectorization (-no-vec)
   b. Try auto-vectorization (-xMIC-AVX512)
   c. Add omp simd instructions
   d. Re-factor code to allow for better vectorization (e.g. veclen, loop re-ordering)
2. Datatype transformations (AoS -> SoA -> AoSoA -> ???)
3. Use the tools
4. Use performance libraries (mkl)

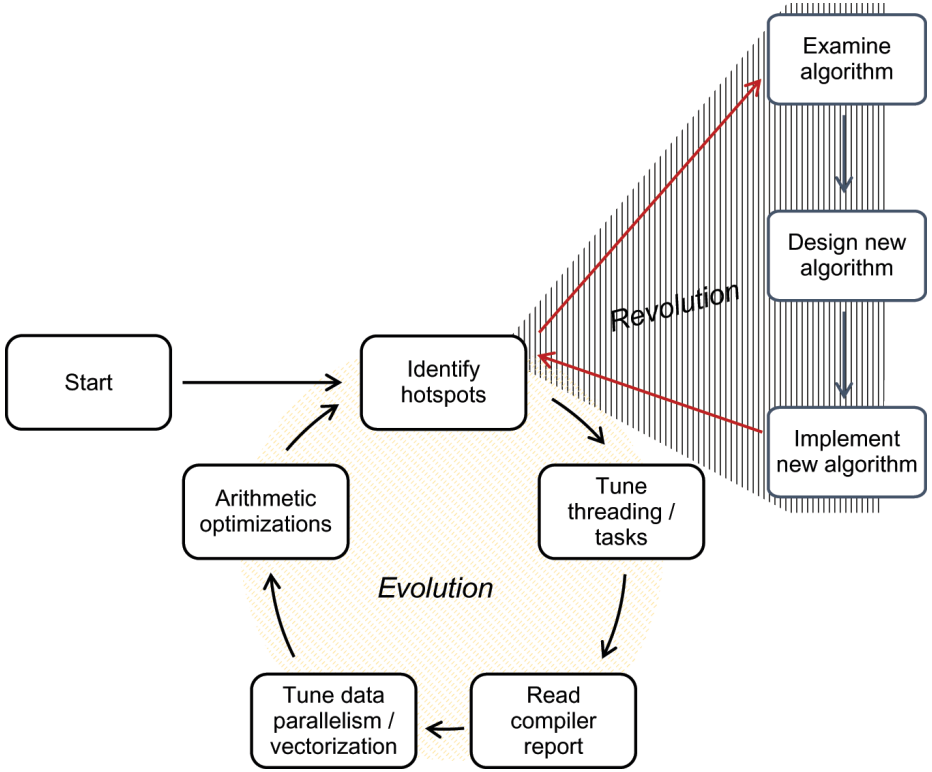
OPTIMIZATION PROCEDURE

Hierarchy of Programming Models/Techniques



OPTIMIZATION PROCEDURE

Lifecycle as an iterative process



How much is enough?

- Focus efforts on high-cost kernels
- Compare performance to roofline

OPTIMIZATION PROCEDURE

Intel-provided tools

- Compiler Reports
  - Add -qopt-report={1-5} to compile line
  - Provides text file with different levels of depth

```
LOOP BEGIN at ../../../../XGC_core/vec/search_vec.F90(53,3)
   remark #15542: loop was not vectorized: inner loop was already vectorized

   LOOP BEGIN at ../../../../XGC_core/vec/search_vec.F90(77,9)
      remark #25045: Fused Loops: ( 77 77 )

      remark #15388: vectorization support: reference at (77:9) has aligned access
      remark #15388: vectorization support: reference psi(:) has aligned access
      remark #15388: vectorization support: reference psi(:) has aligned access
      remark #15388: vectorization support: reference at (77:9) has aligned access
      remark #15305: vectorization support: vector length 4
      remark #15427: loop was completely unrolled
      remark #15309: vectorization support: normalized vectorization overhead 0.444
      remark #15301: FUSED LOOP WAS VECTORIZED
      remark #15448: unmasked aligned unit stride loads: 2
      remark #15449: unmasked aligned unit stride stores: 2
      remark #15475: --- begin vector cost summary ---
      remark #15476: scalar cost: 13
      remark #15477: vector cost: 2.250
      remark #15478: estimated potential speedup: 4.000
      remark #15488: --- end vector cost summary ---
   LOOP END
```

OPTIMIZATION PROCEDURE

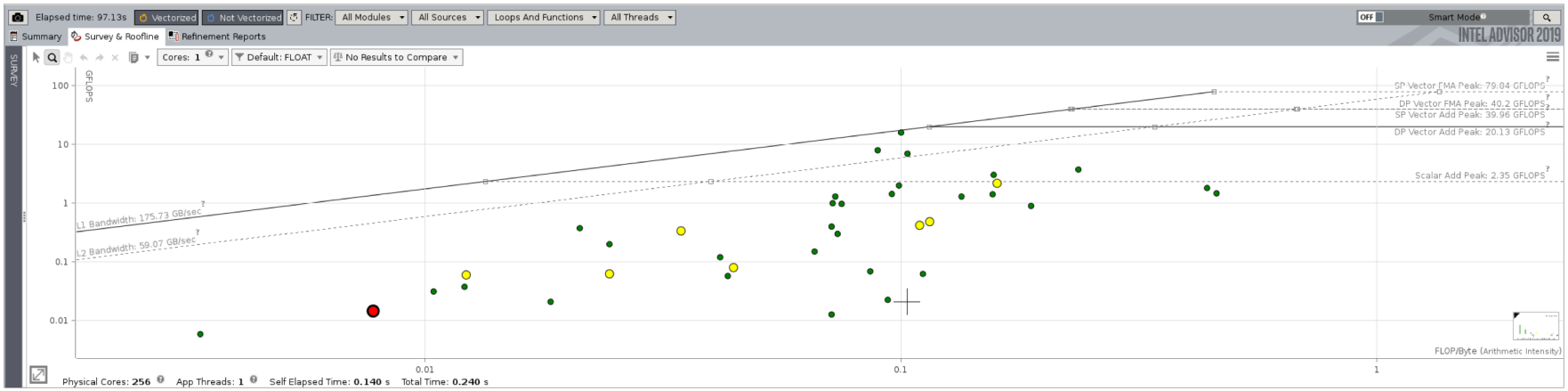
Intel-provided tools

```
vec: do iv=1,veclen
  if (mask(iv)) then

    ! input psi not precomputed
    psi(4) = psi_in(iv)
    !psi(4) = psi_interpol(x(1,iv), x(2,iv), 0, 0)

    do i=1, 3
      nd = grid%nd(i,itr(iv))
      psi(i) = grid%psi(nd)
    enddo
    ! find vertex where psi differs, if all differ, use old method
    psi_diff = 0
    if (abs(psi(1)-psi(2)) .lt. eps1) then
      psi_diff = 3
    elseif (abs(psi(1)-psi(3)) .lt. eps1) then
      psi_diff = 2
    elseif (abs(psi(2)-psi(3)) .lt. eps1) then
      psi_diff = 1
    endif
    if(psi_diff > 0 .and. psi(4) .lt. sml_outpsi) then
      a = MOD(psi_diff, 3) + 1 ! other two vertices
      b = MOD(psi_diff + 1, 3) + 1

      psi(:) = abs(psi(:) - psi(psi_diff))
      p_temp = psi(4) / psi(a)
      p_temp = min(p_temp,1D0)  !! Limit p_temp to 1
      p(psi_diff,iv) = 1D0 - p_temp !new coordinate based on psi
      t_temp = p(a,iv) + p(b,iv)
      ! adjust other coordinates for change in p(psi_diff)
      p(a,iv) = p_temp * p(a,iv) / t_temp
      p(b,iv) = p_temp * p(b,iv) / t_temp
    endif
  endif
enddo vec
```

OPTIMIZATION PROCEDURE

Intel-provided tools

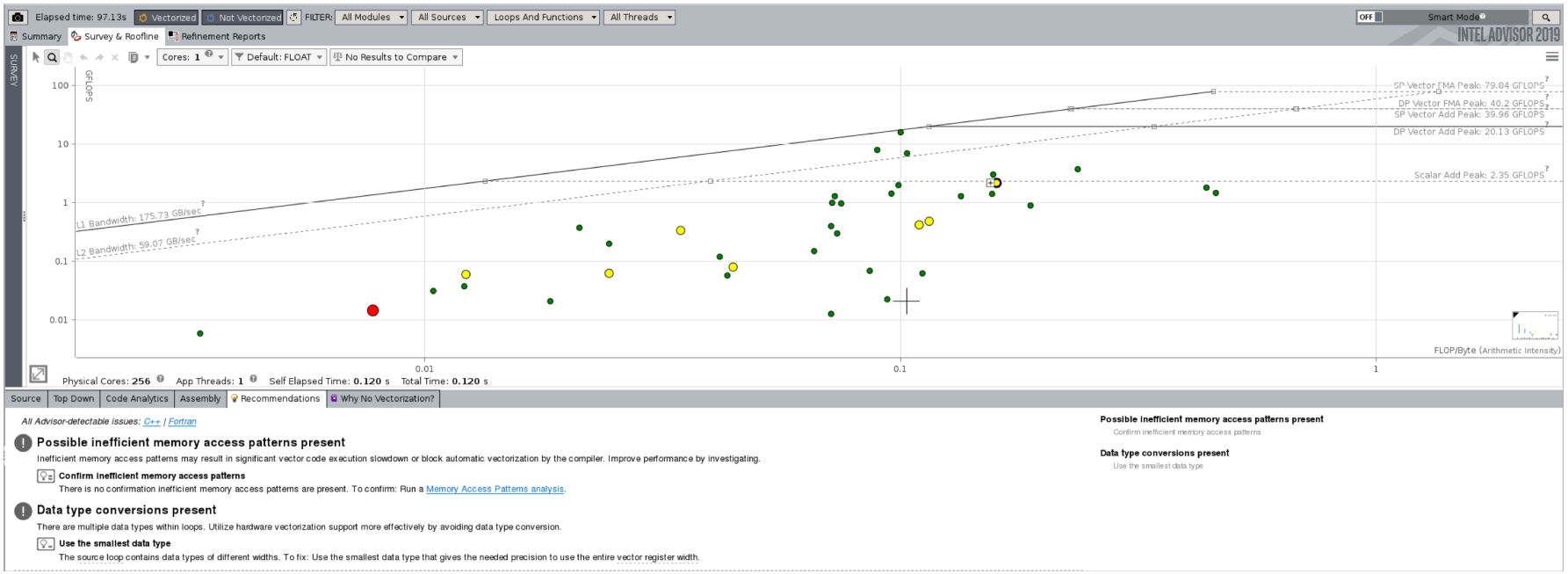
- Roofline
  - Collects data during runtime
  - Provides information on specific loop/functions
  - Informs user of proximity to hardware limits



OPTIMIZATION PROCEDURE

Intel-provided tools

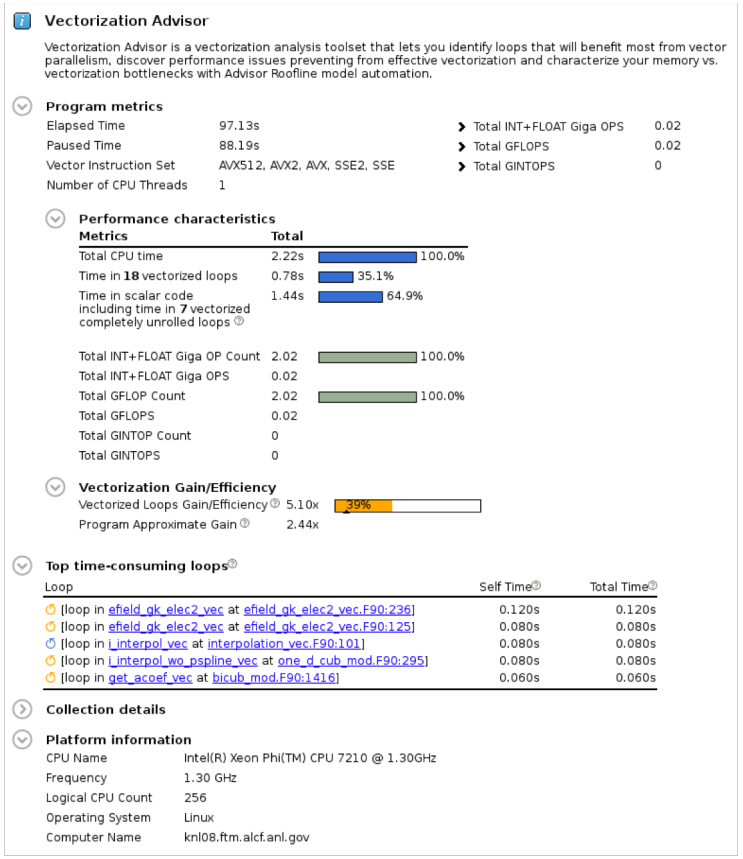
- Roofline



OPTIMIZATION PROCEDURE

Intel-provided tools

- Advisor
  - Provides more metrics
  - Helps to zero in on problem areas



Vectorization Advisor

Vectorization Advisor is a vectorization analysis toolset that lets you identify loops that will benefit most from vector parallelism, discover performance issues preventing from effective vectorization and characterize your memory vs. vectorization bottlenecks with Advisor Roofline model automation.

Program metrics

Elapsed Time	97.13s	Total INT+FLOAT Giga OPS	0.02
Paused Time	88.19s	Total GFLOPS	0.02
Vector Instruction Set	AVX512, AVX2, AVX, SSE2, SSE	Total GINTOPS	0
Number of CPU Threads	1		

Performance characteristics

Metrics	Total	
Total CPU time	2.22s	100.0%
Time in **18** vectorized loops	0.78s	35.1%
Time in scalar code including time in **7** vectorized completely unrolled loops	1.44s	64.9%
Total INT+FLOAT Giga OP Count	2.02	100.0%
Total INT+FLOAT Giga OPS	0.02	
Total GFLOP Count	2.02	100.0%
Total GFLOPS	0.02	
Total GINTOP Count	0	
Total GINTOPS	0	

Vectorization Gain/Efficiency

Vectorized Loops Gain/Efficiency	5.10x	39%
Program Approximate Gain	2.44x	

Top time-consuming loops

Loop	Self Time	Total Time
[loop in efield_gk_elec2_vec at efield_gk_elec2_vec.F90:236]	0.120s	0.120s
[loop in efield_gk_elec2_vec at efield_gk_elec2_vec.F90:125]	0.080s	0.080s
[loop in i_interpol_vec at interpolation_vec.F90:101]	0.080s	0.080s
[loop in i_interpol_wo_pspline_vec at one_d_cub_mod.F90:295]	0.080s	0.080s
[loop in get_acoef_vec at bicub_mod.F90:1416]	0.060s	0.060s

Collection details

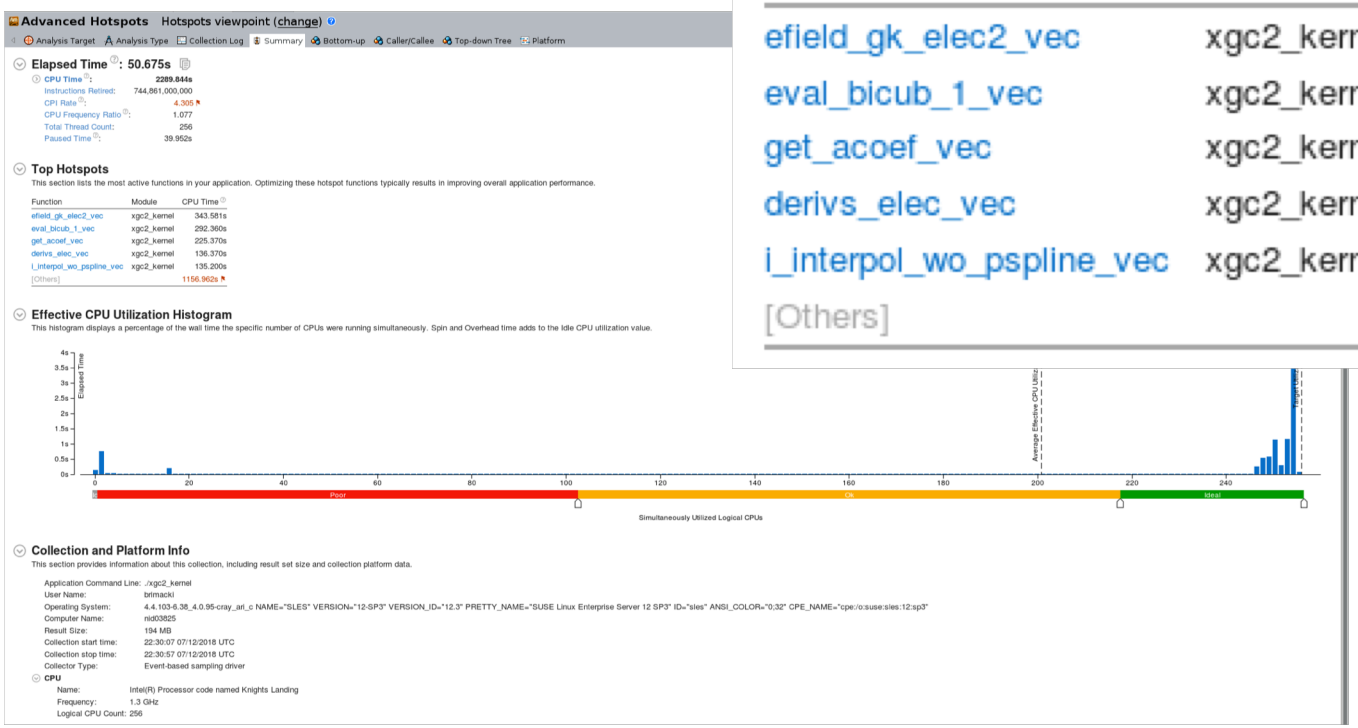
Platform information

CPU Name	Intel(R) Xeon Phi(TM) CPU 7210 @ 1.30GHz
Frequency	1.30 GHz
Logical CPU Count	256
Operating System	Linux
Computer Name	knl08.ftm.alcf.anl.gov

OPTIMIZATION PROCEDURE

Intel-provided tools

- Amplifier
  - Identifies hotspots (expensive)
  - Characterizes multi-threading
  - Measures overall performance (Advisor looks at vectorization)



Function	Module	CPU Time
efield_gk_elec2_vec	xgc2_kernel	343.581s
eval_bicub_1_vec	xgc2_kernel	292.360s
get_acoef_vec	xgc2_kernel	225.370s
derivs_elec_vec	xgc2_kernel	136.370s
i_interpol_wo_pspline_vec	xgc2_kernel	135.200s
[Others]		1156.962s

OPTIMIZATION PROCEDURE

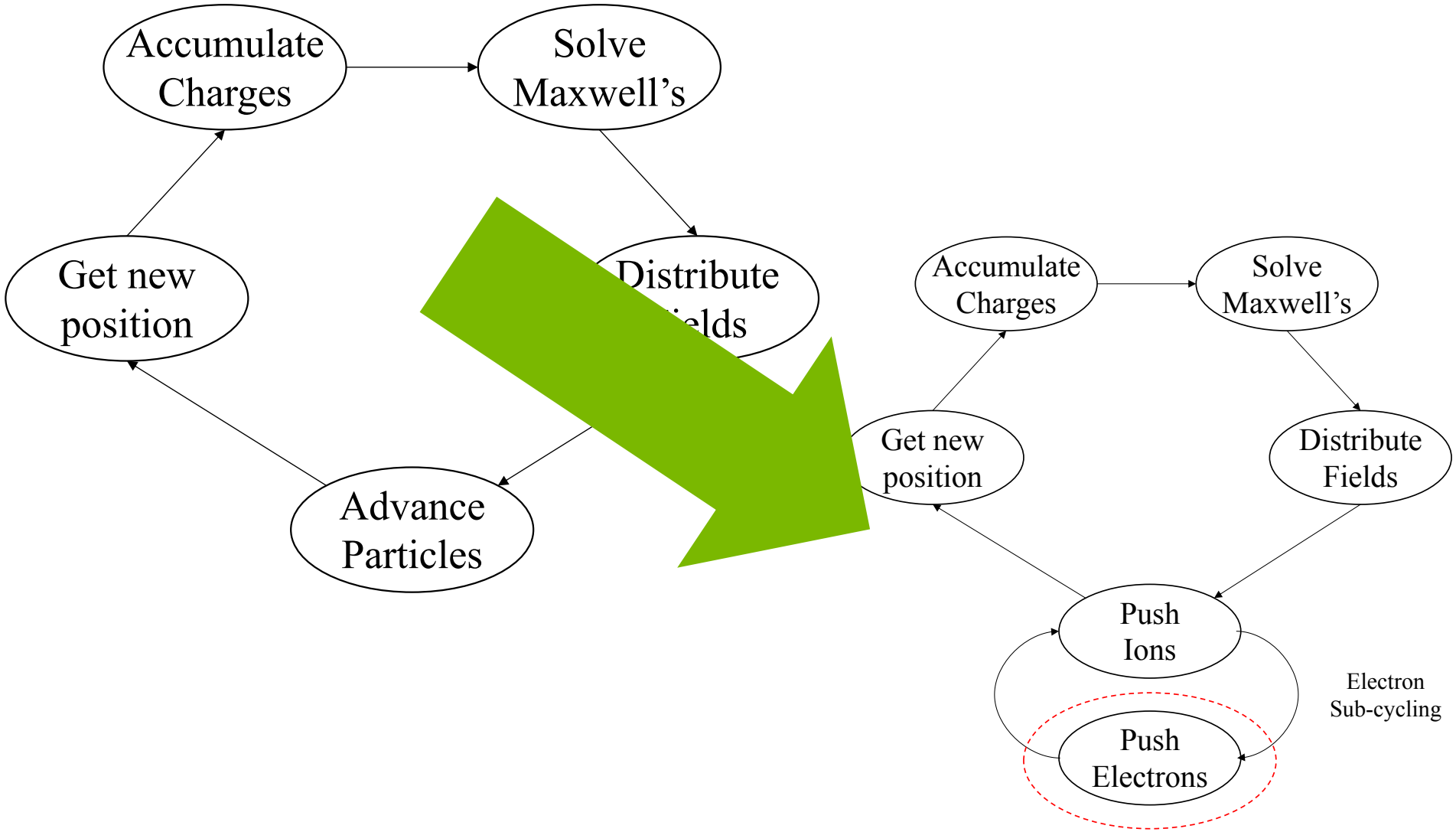
Other opportunities

- MPI Performance Snapshot
- Scaling studies
- Only pass through data needed
- Write clean code
- Develop faster algorithms
- Use more computers
- Try NVIDIA's tools

KNL OPTIMIZATION RESULTS

KNL OPTIMIZATION RESULTS

Electron Push Kernel



KNL OPTIMIZATION RESULTS

Benchmark Problem

- $1/500^{th}$ of toroidal transit time, 4mm mesh size
- Meshing: 75k gridpoints, 150k Δ's in total mesh
- Marker particles: 12.8 million, 800k/MPI rank, 50k/thread
- Tanh equilibrium profile
- δ-f formulation of XGC1
- R = {1, 2.4} m; Z = {-0.67, 0.67} m
- Single-node optimization
- Threads: 256 (16 threads/rank, 16 ranks/node)
- AVX-512 instruction set
- 192 GB DDR4 RAM/node, 16 GB MCDRAM, 300-450 GB/s bandwidth (i.e. Theta)

OPTIMIZATION RESULTS

Base case

Don’t forget the easy stuff!

Compiler flags	O3	O2	O1
512	8.8	8.87	32.77
no_vec	12.42	12.65	32.79
no_simd	8.87	8.89	32.81
no_simd_no_vec	12.42	12.67	32.81

OPTIMIZATION RESULTS

Data structure – A variety of possibilities

AoS Snippet:	SoA Snippet:
type fld_vec(veclen)	type fld_struct
real(8) r, z, phi	real(8), dimension(veclen), allocatable :: r, z, phi
real(8) B_r, B_Φ, B_z	real(8), dimension(veclen), allocatable :: B_r, B_Φ, B_z

SoAoA Snippet:	AoSoA Snippet:
type fld_struct	type fld_veclen
real(8), dimension(3, veclen), allocatable :: rpz	real(8), dimension(array_size), allocatable :: r(veclen), z(veclen), phi(veclen)
real(8) dimension(3,veclen), allocatable :: bvec	real(8), dimension(array_size), allocatable :: B_r(veclen), B_Φ(veclen), B_z(veclen)

OPTIMIZATION RESULTS

Data structure – Results

Table 1: Breakdown of timing by subroutine

Category	FLD PTL SoA SoA	FLD PTL SoA AoS	FLD PTL AoS SoA	FLD PTL AoS AoS	FLD PTL SoAoA AoS
Sort	0.8571 ± 0.0196	0.8588 ± 0.0219	0.8524 ± 0.0140	0.8477 ± 0.0109	0.8573 ± 0.0060
Scatter	0.3621 ± 0.0972	0.4167 ± 0.2019	0.4219 ± 0.1255	0.4508 ± 0.1848	0.3078 ± 0.0666
Gather	0.3570 ± 0.0093	0.3525 ± 0.0083	0.3491 ± 0.0107	0.3578 ± 0.0073	0.3562 ± 0.0093
Time-step	8.2644 ± 0.0709	8.3310 ± 0.0371	10.3138 ± 0.0708	10.4126 ± 0.0823	11.2493 ± 0.0726
Total	9.8405 ± 0.1970	9.9590 ± 0.2962	11.9372 ± 0.2210	12.0689 ± 0.2853	12.7706 ± 0.1545

Table 2: Ranking of data structure choices

FLD/PTL	Time	Speed up
SoA/SoA	9.8405 ± 0.1970	18.46%
SoA/AoS	9.9590 ± 0.2962	17.48%
AoS/SoA	11.9372 ± 0.2210	1.09%
AoS/AoS	12.0689 ± 0.2853	0.00%
SoAoA/AoS	12.7706 ± 0.1545	-5.81%

Conclusion: Structure of Arrays is faster by a large amount.

OPTIMIZATION RESULTS

Memory Allocation

Static (Stack)

main function

for all threads

for all particles

for all sub-cycles

static allocation

push particles

static deallocation

Dynamic (Heap)

main function

dynamically allocate

for all threads

for all particles

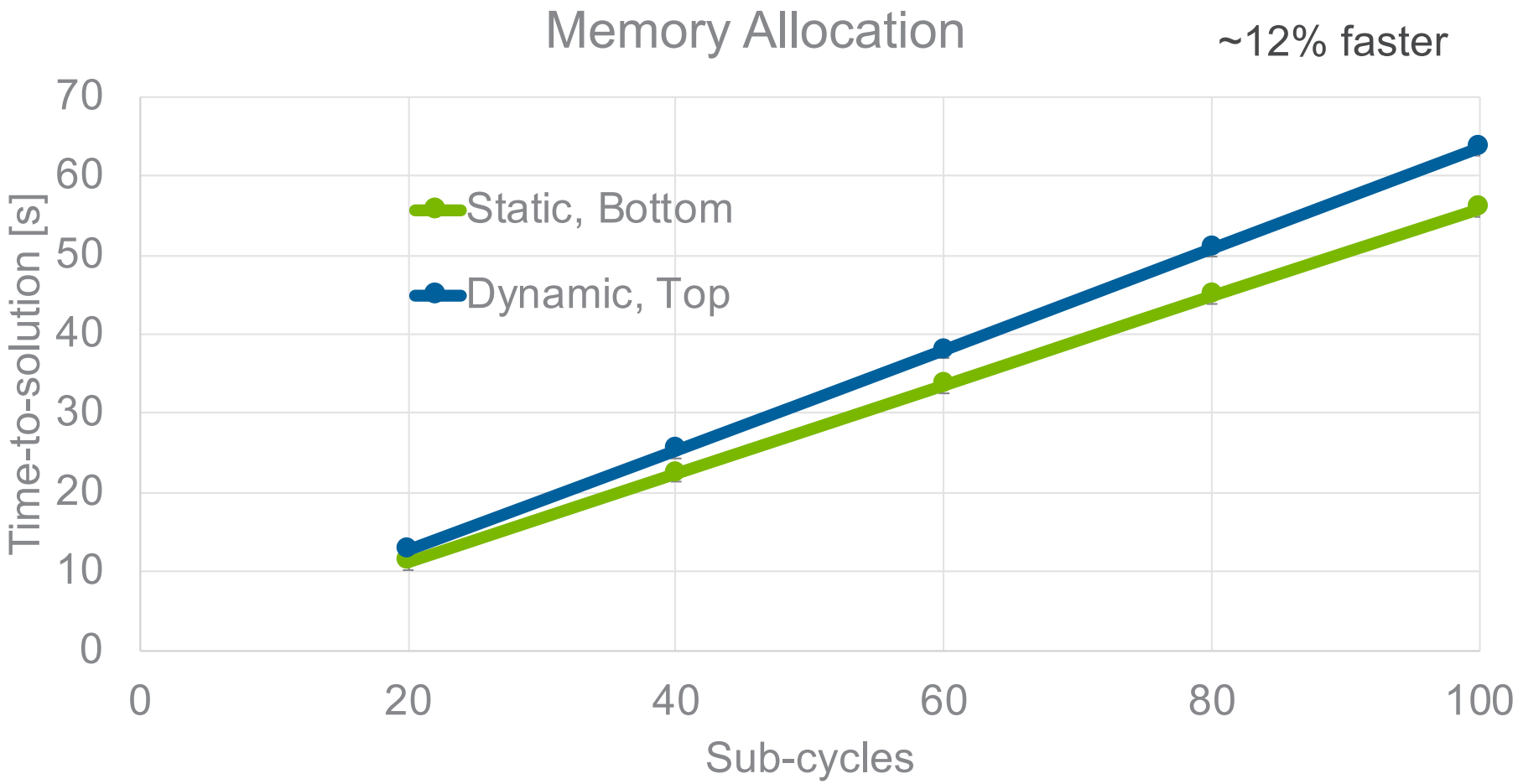
for all sub-cycles

push particles

dynamically deallocate

OPTIMIZATION RESULTS

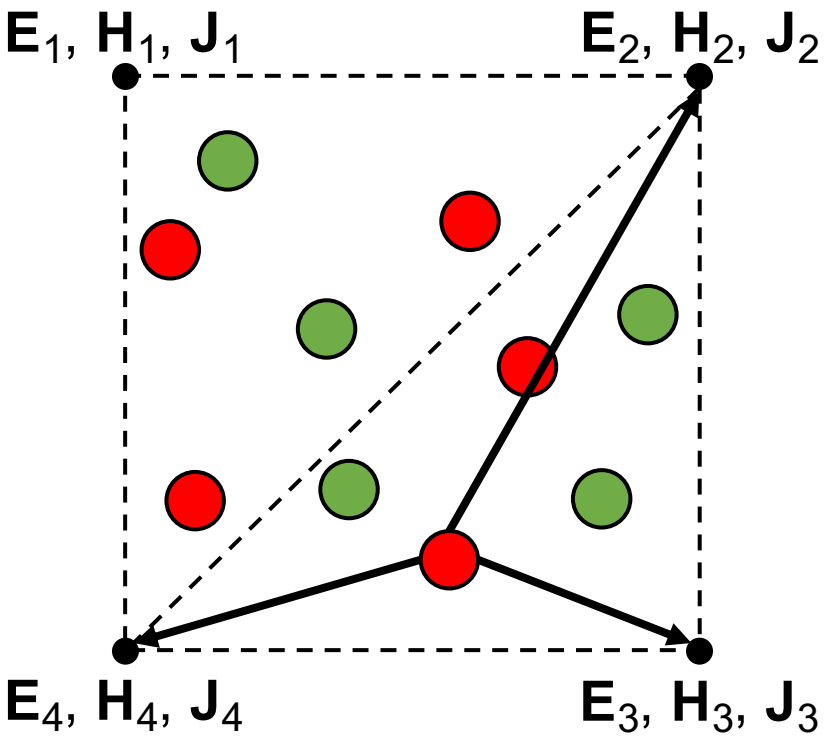
Memory Allocation – Results



OPTIMIZATION RESULTS

Vectorized search

```
subroutine chargee_search_index_local_vec(grid,psn,sp,i_beg,i_end,ith,veclen_max)
  use grid_class
  use psn_class
  use ptl_module
  use sml_module
  use fld_vec_module
  implicit none
  type(grid_type):: grid
  type(psn_type) :: psn
  type(species_type) :: sp
  !
  integer :: veclen_max
  integer :: i, i_beg, i_end, ip, ith, iblock_beg, veclen
  integer :: iv
  integer, dimension(veclen_max) :: itr
  real (8), dimension(veclen_max) :: phi_mid, phi
  real (8), dimension(2,veclen_max) :: x, xff
  real (8), dimension(3,veclen_max) :: p
  logical, dimension(veclen_max) :: mask
#ifdef XGCA
  real (8), external :: psi_interpol
  real (8), dimension(veclen_max) :: psi

  psi = 0D0
#endif


  x = 0D0
  phi = 0D0
  phi_mid = 0D0
  xff = 0D0
  p = 0D0
  itr = -1
  do iblock_beg = i_beg,i_end,veclen_max

     veclen = min(veclen_max,i_end - iblock_beg + 1)

     particleLoop: do iv=1,veclen

        i = iblock_beg + iv - 1

        if(sp%ptl(i)%gid>0)  then

           ! get proper toroidal angle index and weight
           x(1,iv)=sp%ptl(i)%ph(1)
           x(2,iv)=sp%ptl(i)%ph(2)
           phi(iv)=sp%ptl(i)%ph(3)
           phi_mid(iv)=(real(floor(phi(iv) * grid%inv_delta_phi),8) + 0.5D0) * grid%delta_phi
#ifdef XGCA
           psi(iv)=psi_interpol(x(1,iv),x(2,iv),0,0)
#endif

        end if
     end do particleLoop
```



OPTIMIZATION RESULTS

Vectorized search

```
     call field_following_pos2_vec(x(:,1:veclen),phi(1:veclen),phi_mid(1:veclen),xff(:,1:veclen),veclen)
#endif
#ifdef XGCA
     xff = x
#endif

     ! find triangle
#ifdef NO_SEARCH_TR_CHECK
     mask = .true.
     call search_tr2_vec(grid,xff(:,1:veclen),itr(1:veclen),p(:,1:veclen),veclen,mask(1:veclen))
#ifdef XGCA
    if (sml_tri_psi_weighting) then
      mask = (itr .gt. 0)
      call t_coeff_mod_vec(grid,xff(:,1:veclen),psi(1:veclen),itr(1:veclen),p(1:veclen),veclen,mask(1:veclen))
    endif
#endif
```

1. Compilers, like computers, don't know anything. The developer needs to instruct the compiler.
2. These two changes (algorithmic, less data) result in ~30% performance improvement in search; ~3% of whole code.

OPTIMIZATION RESULTS

To-do List

- More in-depth analysis of Advisor reports.
- More “limited data” changes.

AoSoA Snippet:
type fld_vec(array_size)
real(8), dimension(veclen), allocatable :: r, z, phi
real(8), dimension(veclen), allocatable :: B_r, B_Φ, B_z

Current algorithm:
1. Split
2. Search
3. Sort
4. Sub-cycle

Proposed algorithm:
1. Split
2. Search
3. Sort
4. Sub-cycle
    1. Sort some sub-cycles

AURORA

Argonne NATIONAL LABORATORY

EXASCALE (I.E. AURORA) COMPUTING

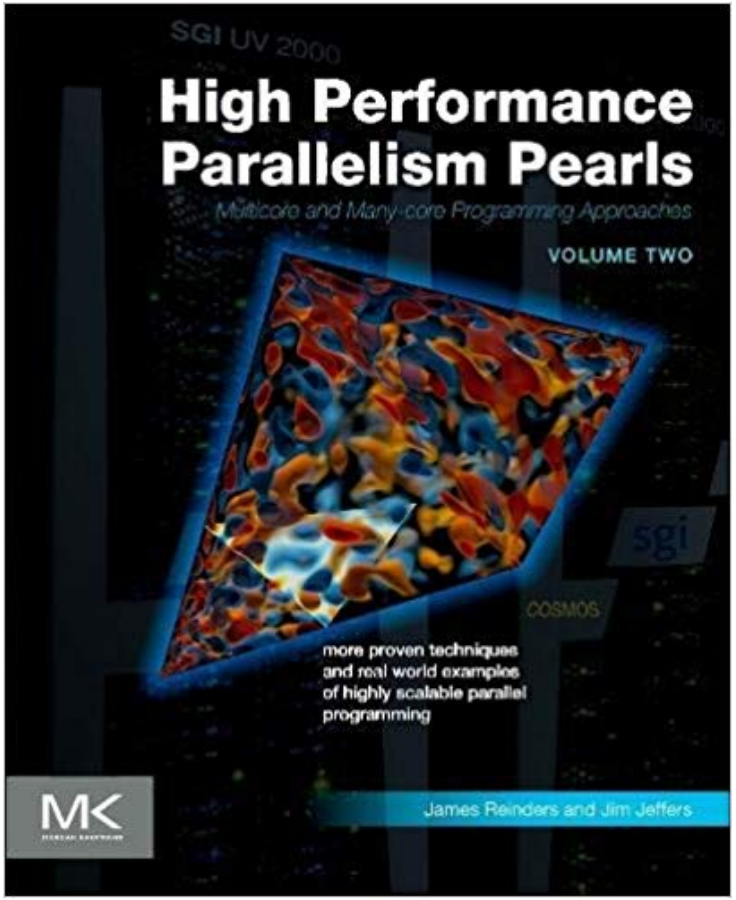
Hot takes

- Understanding of architecture remains important
- Intel KNL serves as a useful basis for future systems
- Lots of folks at Intel interested in making applications work
- Speak to my lawyer (Timothy J. Williams) for more information

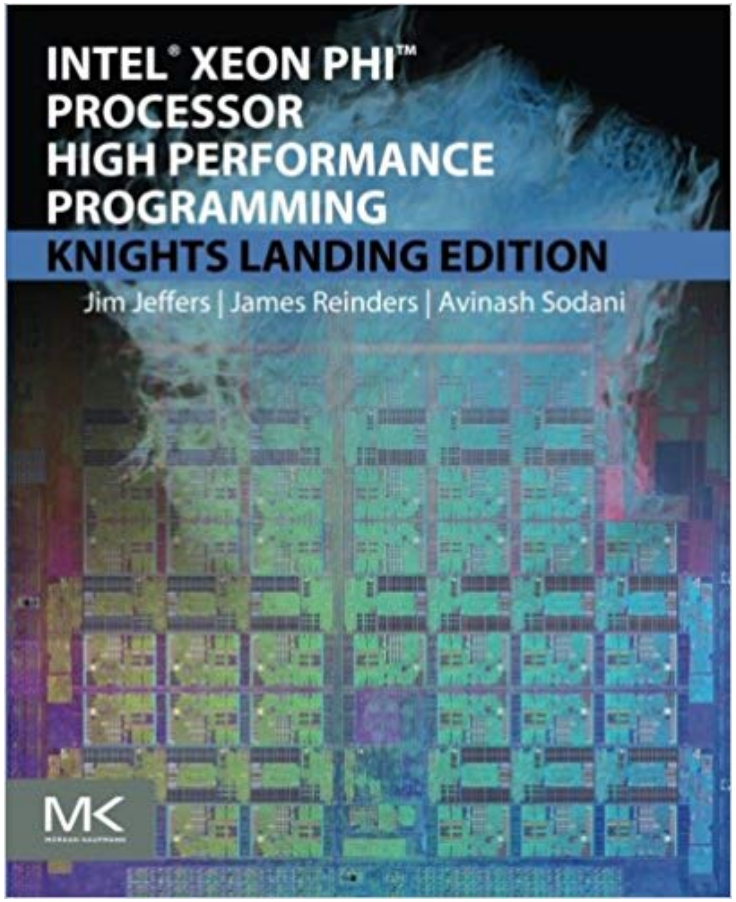
EXTRA RESOURCES

TEXTBOOKS

lotsofcores.com



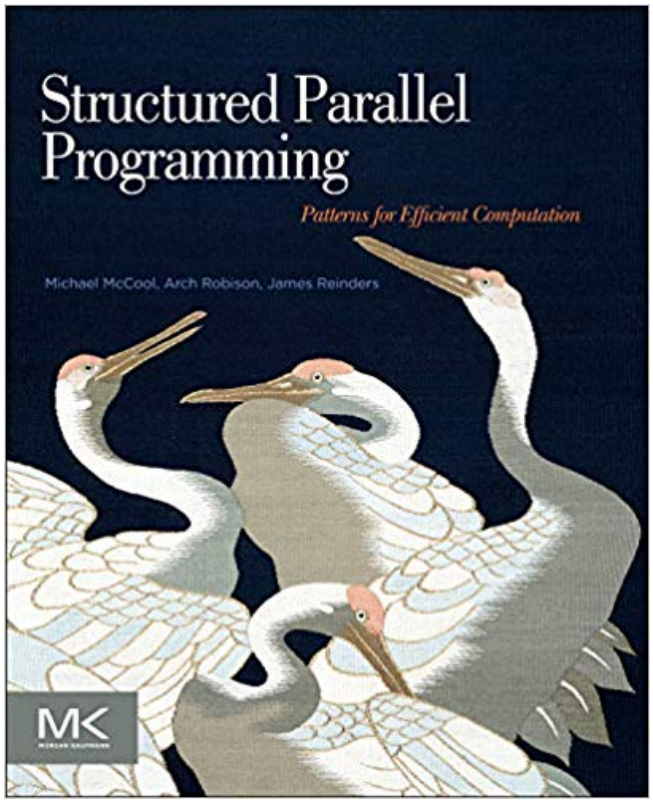
Jeffers, Jim, and James Reinders. *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches*. Morgan Kaufmann, 2015.

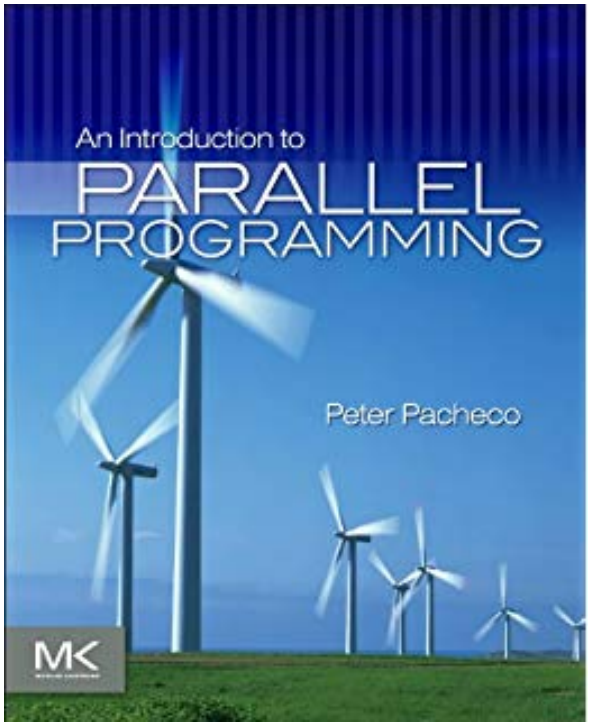


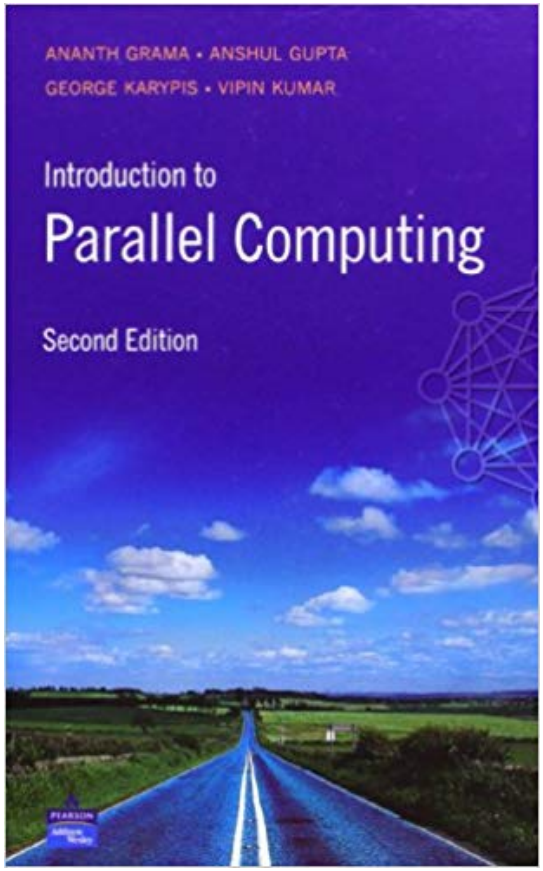
Jeffers, James, James Reinders, and Avinash Sodani. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann, 2016.

MORE RESOURCES

- ALCF, OLCF, NERSC User Guides
- http://software.intel.com/moderncode
- http://lotsofcores.com/sportscar







THANK YOU!!!