

# Advanced OpenMP Constructs, Tuning, and Tools at NERSC



THE UNIVERSITY OF Ahana Roy Choudhury<sup>1</sup>, Yun (Helen) He<sup>2</sup>, and Alice Koniges<sup>2</sup> ALABAMA AT BIRMINGHAM <sup>1</sup>Computer & Information Sciences Department, University of Alabama at Birmingham Knowledge that will change your world <sup>2</sup>NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA

## Abstract

NERSC's next generation supercomputer systems, e.g., Cori Phase 2 with KNL (Intel Knights Landing) architecture, have a large number of cores per node, so, it is important to consider advanced OpenMP concepts in order to achieve optimal performance on such systems.

In this project, we have written and implemented code snippets and used them to test a variety of tools for improving OpenMP performance and detailed their usage on various NERSC systems including KNL. We have explored the detection of issues such as false sharing and data races using tools and how they can be resolved. We have also explored advanced OpenMP concepts such as process and thread affinity and nested OpenMP.

## **Questions and Challenges**

# Improving Performance of Sample Codes using Tools (continued)

#### **Threading Design using Intel Advisor**

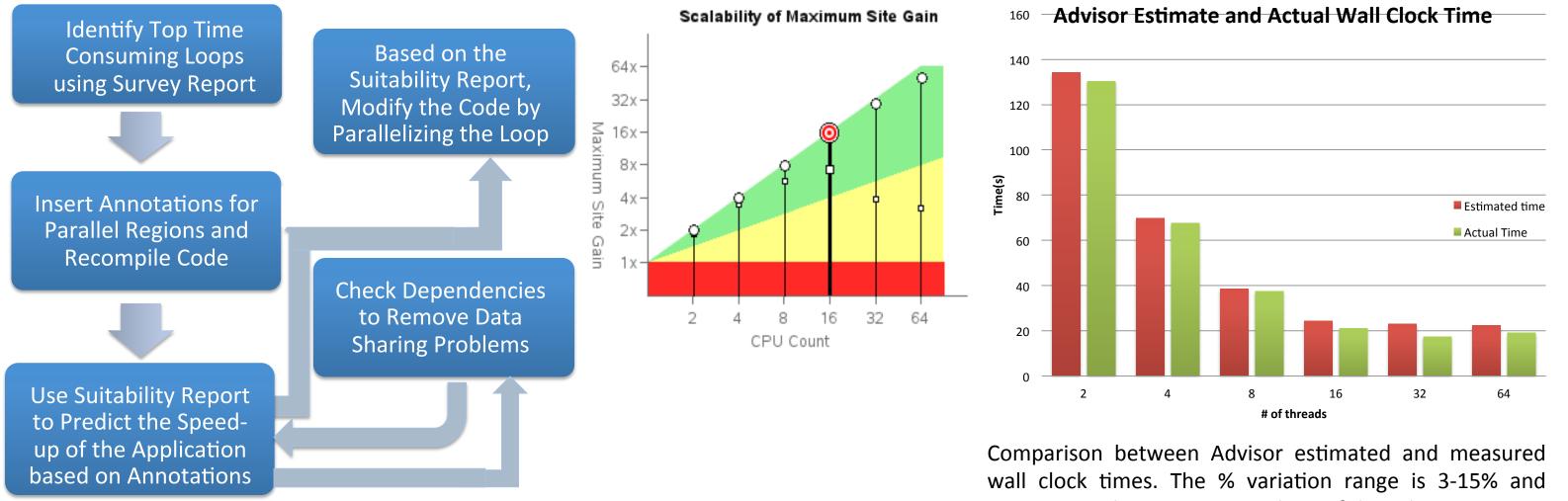
- False sharing occurs when threads on different processors attempt to modify variables that reside on the same cache line.
- It causes performance degradation due to coherence issues and should be avoided.
- Intel VTune Amplifier is a performance analysis tool that helps to analyze the algorithm and identify where and how applications can benefit from available hardware resources.
- To detect false sharing using VTune Amplifier, we wrote a code that causes false sharing, detected the problem and then resolved it using padding. We also noted that some compiler optimization choices remove false sharing.

- Which part of the code can be safely parallelized?
- How to detect and avoid data races?
- How to detect and avoid memory leaks?
- How can tools be used to get suggestions on variable scope and OpenMP compiler directives?
- How to detect top time-consuming loops?
- How to detect and remove false sharing?
- Ensuring desirable process and thread affinity
- Using Nested OpenMP

# Improving Performance of Sample Codes using Tools

#### **Threading Design using Intel Advisor**

- Intel Advisor helps to ensure that Fortran, C and C++ applications take full performance advantage of today's processors.
- Threading Advisor is a threading design and prototyping tool that helps to analyze, design, tune, and check threading design options without disrupting normal development.



# **Before False Sharing Removal**

😔 Elapsed Time 🤄 6.719s 🗊								
<u>CPU Time</u> : 94.372s			72s					
		48.9%						
Grou	Grouping: OpenMP Region / Function / Call Stack							
Op	enMP Region / Function / Call Stack	CPU Time	Memory Bound    L		Lo.	St.	D D D	Average Latency (cycles)
1.	erial - outside any region]	0.323s	56	5.2%	14	16.	0	7
P ma	in\$omp\$parallel:16@unknown:15:25	94.793s	49	9.9%	14	3,2.	0	19
S. Li. ▲	▲ Source			CPU Time				Memory Bound
11	<pre>int A[NUM_THREADS]attribute((aligned(64));</pre>							
12	int n=atoi(argv[1]);							
13	int i;							
<sup>e</sup> 14	int sum=0;							
15	<pre>#pragma omp parallel num_threads(NUM_THREADS) </pre>				This line causes			
16						fals	e sh	aring
17	<pre>int thread = omp_get_thread_num(); </pre>							
18	A[thread] = 0;							
19	<pre>begin=omp_get_wtime(); #ppgmp_get_stime();</pre>							
20	<pre>#pragma omp for schedule(static) for (i=0: i&lt;=n: i++)</pre>				100-		_	10 70/
21				66.490s				49.7%
22 <					83/S			61.3%
23	sum+=A[thread];							
24								
25	1		l					

		After fal			err	101	<i>v</i> al		
		<u>CPU Time</u> <sup>(2)</sup> :		5.153s					
		Memory Bound	®.	2.4%					
	Groupin	g: OpenMP Region / Function / C	all Stack						
7	Openl	MP Region / Function / Call Stack	CPU Time	Memory Bound	Loa.	L	Average Latency (cycles)		
19	▶[Serial	- outside any region]	0.318s 🛛	73.8%	295	6, .	0 9		
	<sup>▶</sup> main\$	omp\$parallel:16@unknown:15:25	4.835s	0.6%	11, 2	2,7.	0 10		
	1	int A[NUM_THREADS]	[16] <u></u> attr	ibute((	alig	ned (	64)));		
	12	<pre>int n=atoi(argv[1]);</pre>							
	13	int i;							
	14	int sum=0;							
	15	#pragma omp parall	el num_thr	eads(NUM_	THRE	ADS)			
	16	{							
<pre>17 int thread = omp_get_thread_num();</pre>									
	18	A[thread][0] = 0;							
	19	begin=omp_get_wti	me();						
	20	<pre>#pragma omp for schedule(static)</pre>							
7%	21	for (i=0; i<=n; i++)							
	22 🤇	A[thread][0] += i%10;							
3%	23	23 #pragma omp atomic							
	24	sum+=A[thread][0];							
	25	}							

# **Process and Thread Affinity**

- Thread affinity binds each process or thread to run on a specific subset of processors, to take advantage of memory locality.
- Improper process/thread affinity could slow down code performance significantly.

#### Using the OMP\_PROC\_BIND environment variable

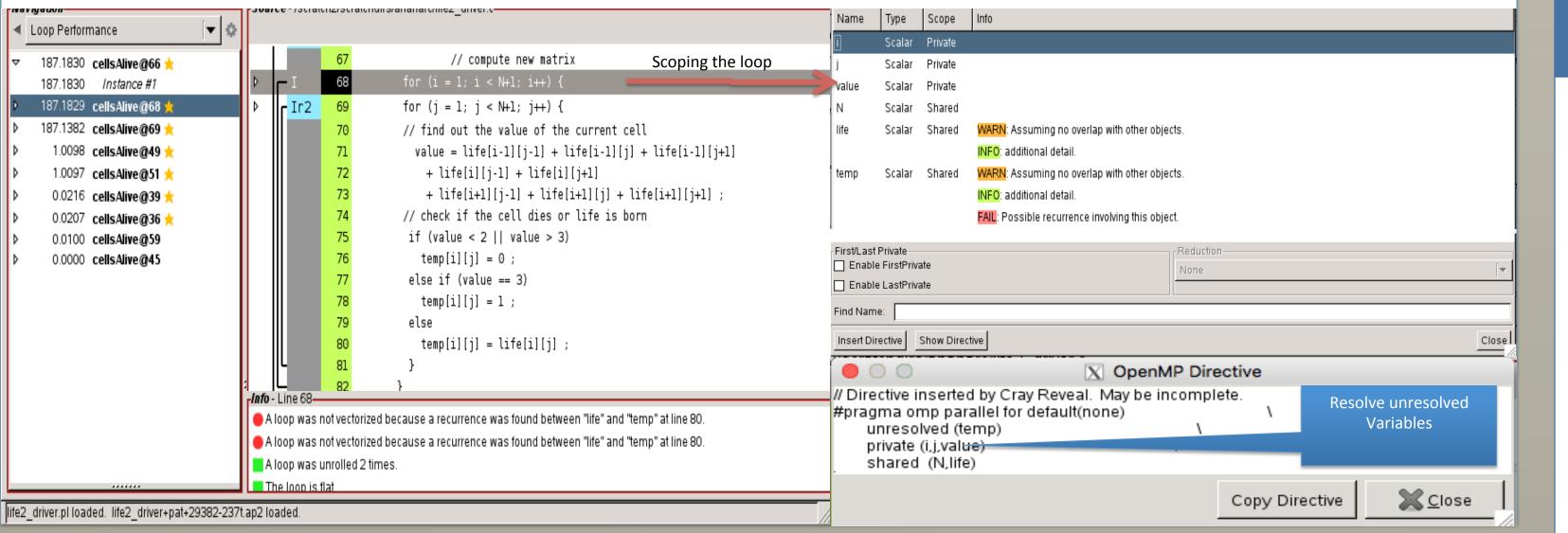
OMP PROC BIND=spread OMP NUM THREADS OMP NUM THREADS

**OMP PROC BIND=close** OMP NUM THREADS OMP NUM THREADS

increases with increasing numbers of threads.

### Parallelizing Codes using Cray Reveal

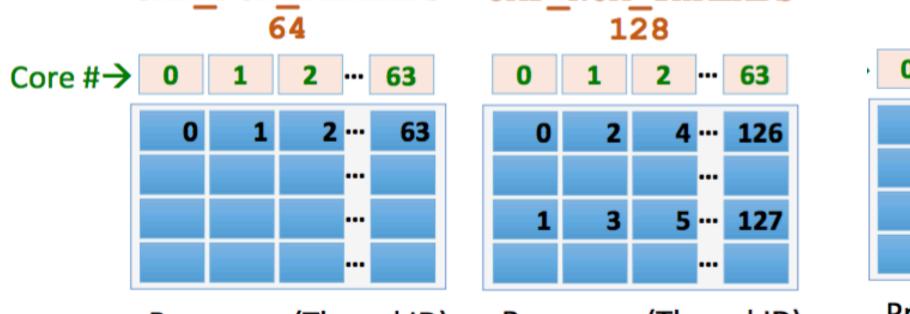
- Reveal is a tool developed by Cray that is part of the Cray Perftools software package.
- Helps to identify top time-consuming loops, dependencies and vectorization.
- Loop scope analysis provides variable scope and compiler directive suggestions for inserting OpenMP.

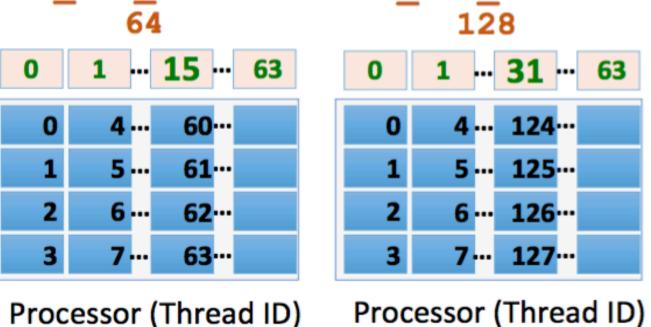


#### **Threading and Memory Error Detection using Intel Inspector**

- Intel Inspector is a dynamic memory and threading error checking tool for users developing serial and multithreaded applications.
  - Example for Detecting Data Pace

#### Example for Detecting Memory Problems

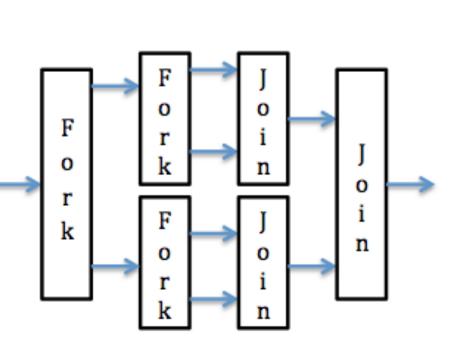




Processor (Thread ID) Processor (Thread ID)

# **Nested OpenMP**

## **Fork and Join Model**

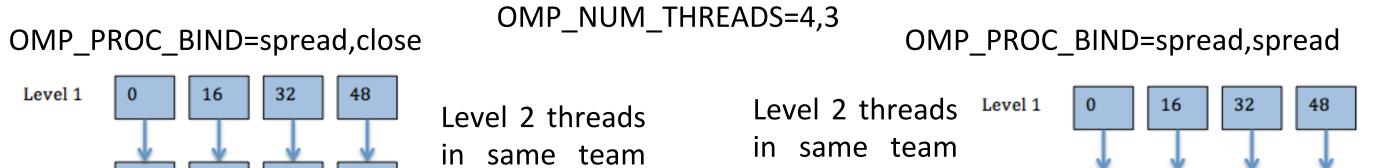


#### When to use Nested OpenMP

#### To achieve more fine-grained thread parallelism:

- When the top level OpenMP loop does not use all available threads
- When multiple levels of OpenMP loops are not easily collapsed
- For certain computation intensive kernels
- For multi-threaded MKL (Intel Math Kernel Library)

## **Thread Affinity for Nested OpenMP**



Example for Detecting Data Race	Example for Detecting Memory Problems
Locate Deadlocks and Data Races Image: Analysis Type Collection Log Summary	Locate Memory Problems Image: Analysis Type Collection Log Summary
Problems         ID▲       ID▲       Type       Sources       Modules       State         P1       Data race       xthi-race.c       xthi-race.impi       New         Data race       xthi-race.impi       New         Data race       xthi-race.impi       New         Data race       xthi-race.impi       New	Problems         ID A       Type       Sources       Modules       Object Size       State         P1       Memory leak       mulmv.c       mulmv.impi       40000       R New         P2       Memory leak       mulmv.c       mulmv.impi       40000       R New         P3       Memory leak       mulmv.c       mulmv.impi       40000       R New         P4       Memory leak       mulmv.c       mulmv.impi       200000000       R New         Description       Source       Function       Module       Object Size       Offset
↓ 1       Implementation       1 of 4 ▶       All       Code Locations: Data race       Implementation	Allocation site mulmv.c:16 main mulmv.impi 40000 14 int n=atoi(argv[1]); 15 int i,j; 16 y=(double *)malloc(n*sizeof(double)); 17 A=(double *)malloc(n*sizeof(double)); 18 x=(double **)malloc(n*sizeof(double));
<pre>52 (void)sched_getaffinity(0, sizeof(coremask), &amp;core 53 cpuset_to_cstr(&amp;coremask, clbuf); 54 global_counter++; 55 #pragma omp barrier 56 printf("Hello from level 1: rank= %d, thread level</pre> xthi-race.impi!main - xthi-race.c:54 xthi-race.impi!main - xthi-race.c	Memory Used by Analysis Tool and Target Application Current memory usage (updated every second): 545 MB
Write       xthi-race.c:54       main       xthi-race.impi         52       (void)sched_getaffinity(0, sizeof(coremask), &core       xthi-race.impi!main - xthi-race.c:54         53       cpuset_to_cstr(&coremask, clbuf);       xthi-race.impi main - xthi-race.c:54         54       global_counter++;       stpragma omp barrier         56       printf("Hello from level 1: rank= %d, thread level	272 MB 136 MB 7 Min 3.5 Min now et Point et P

#### Level 2 32 evenly spread Level 2 bind to threads 16 on threads on on same core 112 165 149 133 different cores 160 234 202 218



In order to achieve good performance in OpenMP codes, it is important to use advanced OpenMP concepts like process and thread affinity and nested OpenMP. It is equally crucial to avoid false sharing and over-subscription. We have explored how tools can be used to facilitate the process of tuning OpenMP codes as well as worked on thread and process affinity and nested OpenMP. Future work involves using nested OpenMP to speed up full applications.

#### References

- 1. https://computing.llnl.gov/tutorials/openMP/
- 2. http://www.eweek.com/c/a/Application-Development/Oracle-and-Java-7-The-Top-10-Developer-Features-626145
- 3. Using OpenMP by Barbara Chapman, Gabriele Jost, Ruud van der Pas, The MIT Press, MIT, 2008.
- s://software.intel.com/en-us/get-started-with-advisor-threading-linux
- https://software.intel.com/en-us/intel-inspector-xe
- 6. <a href="https://www.olcf.ornl.gov/wp-content/uploads/2013/02/Cray\_Reveal-HP1.pdf">https://www.olcf.ornl.gov/wp-content/uploads/2013/02/Cray\_Reveal-HP1.pdf</a>

- 7. http://www.nersc.gov/users/computational-systems/cori/application-porting-and-performance/improving-openmpscaling/
- https://software.intel.com/en-us/articles/avoiding-and-identifying-false-sharing-among-threads
- 9. <u>https://software.intel.com/en-us/articles/finding-your-memory-access-performance-bottlenecks</u>
- 10. https://www.nersc.gov/assets/Uploads/Nested-OpenMP-NUG-20151008.pdf
- 11. http://www.nersc.gov/users/training/events/advanced-openmp-training-february-4-2016/
- 12. https://drive.google.com/a/lbl.gov/file/d/0B9D5EnxRqcaZalR5WEh6bkhxNGs/view

250