Optimization Strategies for Materials Science Applications on Cori

An Intel Knights Landing, Many Integrated Core Architecture

Luther D. Martin[1], and Zhengji Zhao[2](Advisor)

[1]Jackson State University, [2]National Energy Research Scientific Laboratory

Abstract

NERSC is preparing for its next petascale system, Cori [1], a Cray XC system based on an Intel KNL MIC Architecture. Cori will be delivered to NERSC in the fall of 2016. Each compute node will have 72 physical cores and four hardware threads each. This is 12x the number of physical cores than the NERSC's current supercomputer, Edison [2], a Cray XC30. Each Cori node will have 96GB DDR4 memory (500MB per core), and has a 16GB high bandwidth on-package memory (HBM). Cori also comes with a 512-bits vector unit (twice as wide as Edison's). Currently most of the applications that are running on Edison are pure MPI codes, which will not be optimized to take advantage of the higher on-node parallelism, increased processing power and the HBM. In collaboration with Intel, and Cray, NERSC has developed the optimization strategies [3] to help users to get their applications ready for Cori. MPI+OpenMP has been selected as the parallel programming model for Cori to address the increased on-node parallelism. To take advantage of the larger vector unit, vectorization is an essential optimization for Cori. Exploring ways to make efficient use of the HBM is an important optimization as well. To ease the optimization effort, the use of the profiling tools and libraries are strongly recommended by NERSC. In this poster, we recount the effectiveness of three optimization strategies on the number one production code at NERSC, VASP [4-5], a materials science application code. We focus our optimization effort on the single node optimization, which is critical first step to get application codes to perform on Cori.

Optimization Strategies

We implement three optimization strategies to VASP code:

1. Using Intel profiling tool, VTune [6], identify areas in the code that cost a lot of CPU cycles and parallelize them using OpenMP when possible
2. Using the heap manager, Memkind [7], and the AutoHBW[7] library tool that were developed by Intel, to simulate the performance impact of HBM on the available Ivy Bridge node; using Vtune memory-access analysis, identifying candidate arrays for the HBM.
3. Using Intel Vectorization Advisor [8] to analyze the vectorization efficiency, and identify un-vectorized or under-vectorized hotspot loops, vectorize them where applicable.

Results & Discussion

1. Adding OpenMP

- Identified three hotspots in the code
- OpenMP directives were added in the three loops: VHAMIL, LINBAS, and ORTH1
- Used threaded MKL BLAS routines

OpenMP must be implemented carefully to avoid the overhead cost of forking and joining new threads. This is especially true when linking to libraries that leverage OpenMP. For example, many calls are made to threaded MKL BLAS routines. Placing this calls inside of an outer parallel region allows subsequent parallel regions in the BLAS routines to use a thread from the thread pool created from the outside parallel region. This cut downs the overhead associated with forking new parallel regions

Function / Memory Object / Allocation Stack	CPU Time		Function / Memory Object / Allocation Stack	LLC Miss
__kmp_fork_barrier	934.209s		__kmp_fork_barrier(int, int)	kmp_barrier.cpp:0
[MKL BLAS]@zgemm_omp_driver_pre_avx2	43.818s		mkl_blas_zgemm_omp_driver_pre_avx2	0
__kmp_fork_call	37.835s		__kmp_fork_call	kmp_runtime.c:0
[MKL Iterative Solvers]@compute_bwd_d_c2c_1d_i	31.121s		mkl_dft_avx_compute_bwd_d_c2c_1d_i	0
[MKL Iterative Solvers]@compute_fwd_d_c2c_1d_i	27.270s		mkl_dft_avx_compute_fwd_d_c2c_1d_i	0
MPID_nem_gni_poll	25.151s		MPID_nem_gni_poll	0
LA_Loop4gas_1	16.883s		LA_Loop4gas_1	0
vhamil	14.989s		vhamil	hamil.f90:0
GNI_CqGetEvent	14.825s		GNI_CqGetEvent	0
GNII_DlaProgress	14.795s		GNII_DlaProgress	0
MPIDI_CH3I_Progress	14.704s		MPIDI_CH3I_Progress	0
[MKL BLAS]@trmm_host	13.382s		mkl_blas_trmm_host	0
MPID_nem_gni_check_localCQ	13.056s		MPID_nem_gni_check_localCQ	0
[MKL BLAS]@zaxpy_omp	10.824s		mkl_blas_zaxpy_omp	0

Figure 1. Vtune hotpots analysis shows extremely high overhead when threaded MKL BLAS routine was called form the VASP code.

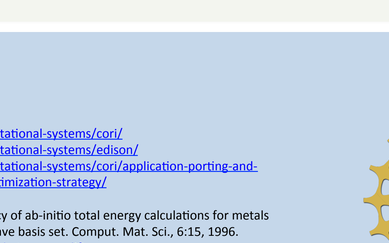

Fig 2. This chart depicts the overhead cost associated with forking and joining threads for two routines: ORTH1 and LINBAS. The results show that it's better to create a global parallel region which maintains a pool of threads than forking new threads every time a parallel region is encountered.

Fig 3. OpenMP/MPI scaling test Result

Fig 4. OpenMP threads scaling results

Fig 5. On package, High-bandwidth Memory

Fig 6. Edison memory architecture used to simulate Cori's HBM

2. Move large arrays to HBM

- Move arrays to on package memory explicitly in code or set array size threshold and allow Intel's *AutoHBW* tool to move arrays automatically
- Identify candidate arrays

Grouping: Function / Memory Object / Allocation Stack					
Function / Memory Object / Allocation Stack	CPU Time	Loads	LLC Miss	Average Latency	Memory Object ...
altstack_sampling_offline	133.119s	3,364,100,920	0	7	
[MKL BLAS]@zgemm_kernel_0	57.430s	1,049,231,476	0	10	
wave.f90:587	0s	13,900,417	0	13	650 MB
allocw ⇐ vamp ⇐ main ⇐ _start	0s	13,900,417	0	13	650 MB
choleski2.f90:121	0s	900,027	0	10	5 MB
orthch ⇐ vamp ⇐ main ⇐ _start	0s	900,027	0	10	5 MB
choleski2.f90:121	0s	900,027	0	11	5 MB
subrot.f90:188	0s	800,024	0	10	5 MB
subrot.f90:188	0s	600,018	0	10	5 MB
choleski2.f90:121	0s	500,015	0	10	5 MB
scala.f90:347	0s	200,006	0	9	5 MB
scala.f90:347	0s	100,003	0	13	5 MB
dfast.f90:448	0s	165,504,965	0	10	3 MB
linbas ⇐ lincom ⇐ eddiag ⇐ elmin ⇐ vamp ⇐ main ⇐ _start	0s	165,504,965	0	10	3 MB
dfast.f90:448	0s	164,404,932	0	10	3 MB
dfast.f90:448	0s	3,600,108	0	9	3 MB
dfast.f90:448	0s	3,600,108	0	10	3 MB
dfast.f90:448	0s	300,009	0	10	2 MB

Fig 7. VTune memory-access analysis can identify source lines and arrays that generate high memory traffic. Once these arrays are identified, allocate these arrays in HBM by adding the Intel compiler directive !DIR$ ATTRIBUTES FASTMEM to the codes

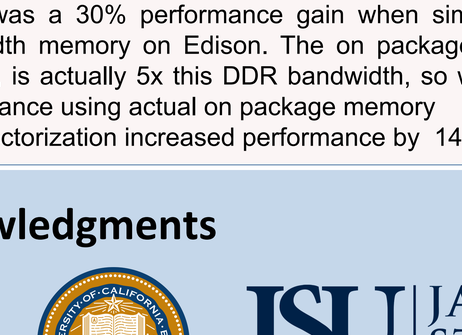

Fig 8. On package, high-bandwidth memory performance

3. Refactor code to allow compiler to easy vectorize

- Use Intel's Advisor tool and compiler opt-reports to identify loops that won't vectorize
- Refactor code to remove if clauses, dependencies, etc. to allow compiler to vectorize loop.

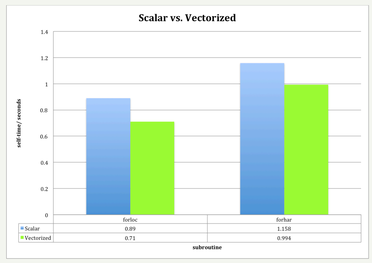

Fig 9. This graph shows two subroutines that contained loops that wouldn't vectorize due to inner-loop dependencies. After removing dependencies, we saw a 20% performance increase in the *forloc* routine and a 14% increase *in forhar* routine.

Conclusions

- OpenMP is the direction to go for highly parallelized architecture. There are notable performance gains when implementing OpenMP around loops.
- Global parallel regions cut down on the overhead cost of forking new parallel regions.
- There was a 30% performance gain when simulating the high-bandwidth memory on Edison. The on package, high-bandwidth on Cori, is actually 5x this DDR bandwidth, so we expect greater performance using actual on package memory
- Loop vectorization increased performance by 14 - 20 percent

Contact

Luther Martin
Jackson State University
Email: luther.d.martin@gmail.com
Phone: 601-927-9322

References

1. https://www.nersc.gov/users/computational-systems/cori/
2. https://www.nersc.gov/users/computational-systems/edison/
3. https://www.nersc.gov/users/computational-systems/cori/application-porting-and-performance/getting-started-and-optimization-strategy/
4. VASP: https://www.vasp.at/
5. G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. Comput. Mat. Sci., 6:15, 1996.
6. https://software.intel.com/en-us/intel-vtune-amplifier-xe
7. https://github.com/memkind/memkind
8. https://software.intel.com/en-us/intel-advisor-xe

Acknowledgments

TRUST Team for Research in Ubiquitous Secure Technology