Power Benchmarking and Compiler Comparison for Intel Xeon Phi KNL Architecture

Dhruv Jain, Chinmay Joshi and Dr. Rong Ge
Clemson University

Dhruv Jain
School of Computing
Clemson University
Clemson, SC USA
djain@clemson.edu

Chinmay Joshi
Electrical and Computer Engineering
Clemson University
Clemson, SC USA
chinmaj@clemson.edu

***Abstract*—Energy efficiency in processors has become a critical issue after the large scale usage of computers worldwide. In this paper we will analyze the energy, power and performance parameters of Intel KNL (Knights Landing) architecture. We investigated difference in above parameters with using different compilers (gcc and icc) in OpenMP benchmark applications. We used RAPL measurements for our study. Understanding power measurements and issues related to it is important to successfully deploy energy efficient high performance computing clusters.**

Keywords—energy; power; benchmark; knl

I. Introduction

The main motivation behind this paper is to study energy, power and performance parameters in Intel KNL processor. Processors are used in data centers and cloud computing which makes it large scale use. The servers need to be always running and require energy all the time so it is important we study the energy efficiency in processors and find ways to optimize them. There is added demand for further processors which makes it even more critical to study the energy impact of modern processors. There are projections that information and communications technologies, including computers and cell phones, will consume 14 percent of worldwide electricity by 2020[1]. The paper focuses on compiler differences between icc and gcc and we found out that icc performed better than gcc. Section II talks about the KNL architecture, Section talks about the various benchmark and monitoring tools we used in this project, Section III and IV are two different experiments performed on KNL and in Section V we will analyze the results of those experiments.

II. KNL Architecture

Knights Landing is a second generation MIC architecture product from Intel. The below screenshot shows the architecture details for KNL. We used Xeon Phi 7210 model to perform experiments. Each core have two 512-bit vector units and support AVX-512 SIMD instructions, specifically the Intel AVX-512 Foundational Instructions (AVX-512F) with Intel AVX-512 Conflict Detection Instructions (AVX-512CD), Intel AVX-512 Exponential and Reciprocal Instructions (AVX-512ER), and Intel AVX-512 Prefetch Instructions (AVX-512PF) [2].

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                256
On-line CPU(s) list:   0-255
Thread(s) per core:    4
Core(s) per socket:    64
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 87
Model name:            Intel(R) Xeon Phi(TM) CPU 7210 @ 1.30GHz
Stepping:              1
CPU MHz:               1045.941
BogoMIPS:              2594.04
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
NUMA node0 CPU(s):     0-255
```

Fig. 1. KNL Architecture - the above screenshot shows various architecture details including number of cores - 64 and number of processors - 256.

III. Benchmarks, Performance Monitoring tool and Energy Values

Initially we started researching on various power benchmarks that we could potentially use on Intel Phi KNC's and KNL's. The SPEC OMP and SPEC HPC Benchmarks suite are not open source so we did not implement those. We also ran Parsec (The Princeton Application Repository for Shared-Memory Computers) Benchmark on KNC Intel Phi on Palmetto but unfortunately we could not collect any power measurements.

A. Likwid

LIKWID Benchmark Suite - Likwid is an open-source

Performance monitoring and benchmarking suite. Likwid works for Intel and AMD processors on the Linux operating system. Likwid consists of the following 12 categories which can be used in command line to produce desired output. It consists of the following:

- *likwid-pin:* pin your threaded application (pthread, Intel and gcc OpenMP to dedicated processors)
- *likwid-bench:* Micro benchmarking platform
- *likwid-features:* Print and manipulate cpu features like hardware prefetchers
- *likwid-genTopoCfg:* Dumps topology information to a file
- *likwid-mpirun*: Wrapper to start MPI and Hybrid MPI/OpenMP applications (Supports Intel MPI, OpenMPI and MPICH)
- *likwid-perfscope:* Frontend to the timeline mode of likwid-perfctr, plots live graphs of performance metrics using gnuplot
- *likwid-agent:* Monitoring agent for hardware performance counters
- *likwid-memsweeper:* Sweep memory of NUMA domains and evict cachelines from the last level cache
- *likwid-setFrequencies:* Tool to control the CPU frequency

Installation steps for likwid:

$ tar -xjf likwid-4.2.tar.bz2
$ cd likwid-4.2
$ vi config.mk (Changes to likwid config - configure build, e.g. change installation prefix)
$ make
$ sudo make install

In this paper we have used the following likwid commands to read energy values and other parameter - *Likwid-perfctr* (Likwid Group - ENERGY). Likwid reads RAPL values for Intel Xeon Phi KNL family (x200) processors.

TABLE I. VARIOUS COUNTERS AND EVENTS

Counter name	Event name	
PWR0	PWR_PKG_ENERGY	PROCESSOR DIE
PWR1	PWR_PP0_ENERGY	PROCESSOR CORE SUBSYSTEM
PWR3*	PWR_DRAM_ENERGY	DIRECTLY ATTACHED DRAM

The above table shows the three RAPL parameters we measured.

Formulas used by various power counters in likwid -
Power = PWR_PKG_ENERGY / time
Power PP0 = PWR_PP0_ENERGY / time
Power DRAM = PWR_DRAM_ENERGY / time

B. CPU Utilization

Script we used to calculate approximate average CPU utilization:

top -b -n2 -p 1 | fgrep "Cpu(s)" | tail -1 | awk -F'id,' -v prefix="$prefix" '{ split($1, vs, ","); v=vs[length(vs)]; sub("%", "", v); printf "%s%.1f%%\n", prefix, 100 - v }' [3]

The script takes a snapshot of results from top -b -n2 -p 1 command.

C. NASA Parallel Benchmarks (NPB)

The NAS Parallel Benchmarks (NPB) are a set of benchmarks targeting performance evaluation of multicore processors.

- *IS* - Integer Sort, random memory access
- *EP* - Embarrassingly Parallel
- *CG* - Conjugate Gradient, irregular memory access and communication
- *MG* - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
- *FT* - discrete 3D fast Fourier Transform, all-to-all communication
- *BT* - Block Tri-diagonal solver
- *SP* - Scalar Penta-diagonal solver
- *LU* - Lower-Upper Gauss-Seidel solver

NPB eight benchmarks are defined in various problem sizes. We used Class S (smaller dataset) for experiment 1. We used Class A, Class B and Class C for Experiment 2 which are much larger datasets increasing in size from Class A to C.

D. Compilers and flags

Intel's x200 architecture supports Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instruction set architecture (ISA) [4].

Intel AVX-512 instruction set architecture (ISA) consists of the following groups:

Intel AVX-512 Foundation instructions (AVX-512F) are the base of Intel AVX-512. They include extensions of the Intel AVX and Intel AVX2 family of SIMD instructions but are encoded using EVEX encoding scheme with support for 512-bit vector registers, up to 32 vector registers in 64-bit mode, and conditional processing using opmask registers. Intel AVX-512 Conflict Detection instructions (AVX-512CD) provide efficient conflict detection to allow more loops to be

vectorized.

Intel AVX-512 Exponential and Reciprocal instructions (AVX-512ER) are designed to provide building blocks for accelerating certain transcendental math computations. Intel AVX-512 Prefetch instructions (AVX-512PF) are new instructions that can be useful for reducing memory operation latency exposure that involve gather/scatter instructions. Intel (AVX-512BW) extend AVX-512 instruction set to cover 8-bit and 16-bit integer operations.

Intel (AVX-512DQ) are new 32-bit and 64-bit AVX-512 instructions for enhancing integer and floating-point operations.

Intel AVX-512 Vector Length Extensions (AVX-512VL) extends most AVX-512 operations to also operate on XMM (128-bit) and YMM (256-bit) registers, instead of only ZMM (512-bit) registers.

AVX-512F, AVX-512CD, AVX-512ER, and AVX-512PF are implemented in the Intel Xeon Phi processor x200 (code named Knights Landing)

ICC provides handy way of enabling these groups with flag -xMIC-AVX512 compiler flag. In GCC, support for AVX-512F, AVX-512CD groups is available with flags -mavx512f and -mavx512cd respectively.

In Experiment-2, we compared GCC version 6.3.0 and compatible version of ICC 17.0.02. in Experiment 1, we used older version of GCC where AVX512 flags were not available.

IV. EXPERIMENT 1

In first experiment we compared GCC version 4.8.5 and ICC version 17.0.02 compatible with GCC 4.8.5. We didn't use any compiler flags on GCC for vector optimizations, as GCC 4.8.5 does not supports these optimizations for x200 KNL architectures. We used class S for NPB benchmark applications.

gcc version 4.8.5 20150623 (Red Hat 4.8.5-11)
Using -fopenmp -O3 -lm

icc version 17.0.2 (gcc version 4.8.5 compatibility)
Using -qopenmp -O3

icc version 17.0.2 (gcc version 4.8.5 compatibility)
Using -qopenmp -O3 -xmic-avx512

GCC 4.8.5 vs ICC NPB Class S

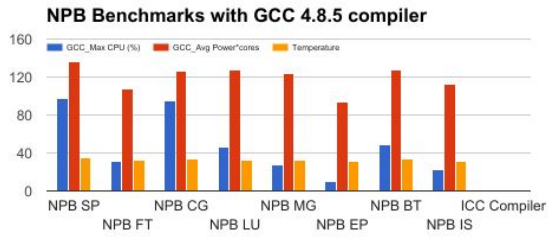

Fig. 2. NPB Benchmark with GCC 4.8.5 Class S

The above graph in Fig. 2 shows that gcc performed better than ICC in max CPU utilization.

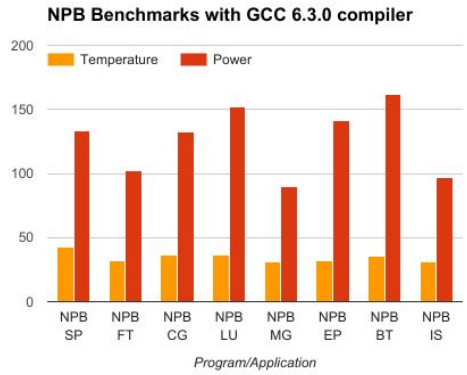

Fig. 3. NPB Benchmark with GCC 6.3.0 Class S

The above graph in Fig. 3 shows gcc power and temperature values on Class S datasets.

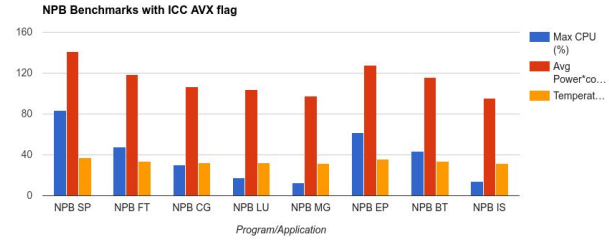

Fig. 4. NPB Benchmark with ICC AVX flag only Class S

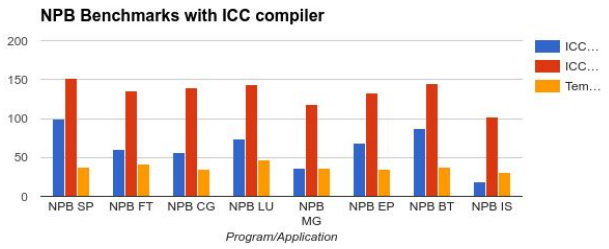

Fig. 5. NPB Benchmark with ICC compiler Class S

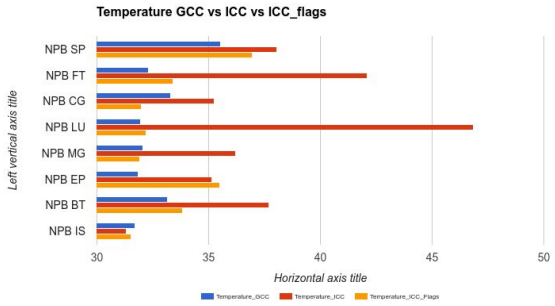

Fig. 6. Temperature comparison Class S

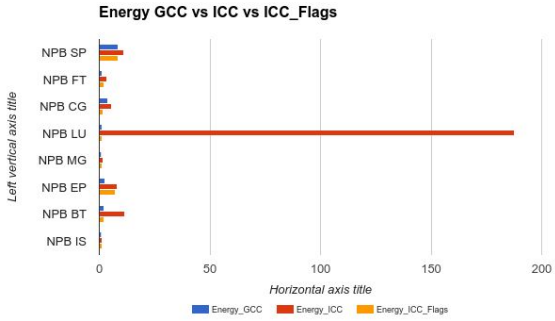

Fig. 7. Energy comparison Class S

From the above charts you can see that when using gcc compiler version 4.8.5, gcc performed better than icc compiler. The worst performance we see here is from icc without any flags.

This result made us curious about the doing more investigations in compiler/compiler flags and energy performance. We did Experiment 2 with using latest versions of GCC and ICC.

V. Experiment 2

In second experiment, we worked on latest versions of GCC and ICC. We used following compilers with compiler options we used:

icc version 17.0.2 (gcc version 6.3.0 compatibility)

icc -qopenmp -O3 -xmic -avx512

gcc version 6.3.0 (GCC)

gcc -fopenmp -O3 -mavx512f -mavx512cd -lm

We used Class A,B,C of NPB benchmarks for comparison. Class A,B,C are built for larger problem datasets and better suitable for vector optimizations.

GCC 6.3 vs ICC NPB Class A, B, C

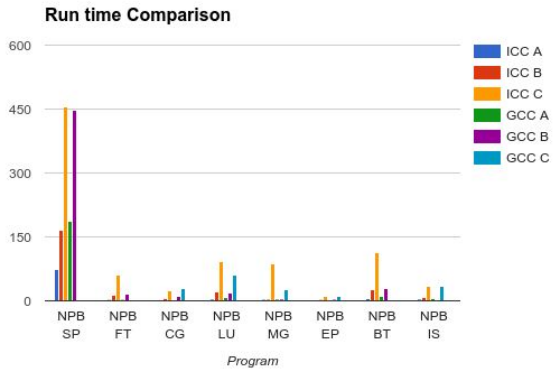

Fig. 8. Runtime comparison GCC vs ICC for Class A, B and C datasets.

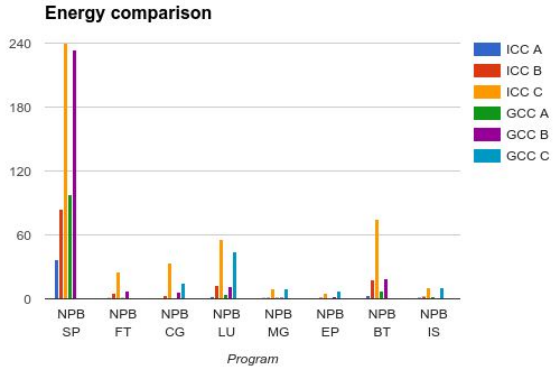

Fig. 9. Energy comparison GCC vs ICC for Class A, B and C datasets.

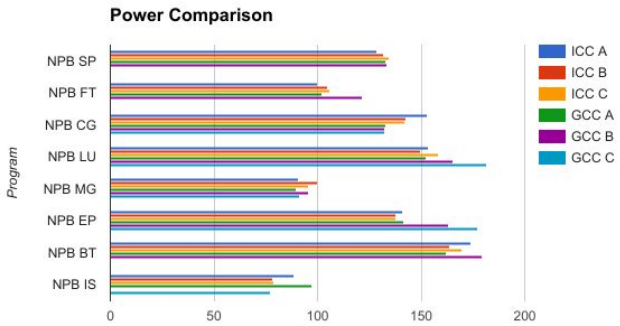

Fig. 10. Power comparison GCC vs ICC for Class A, B and C datasets.

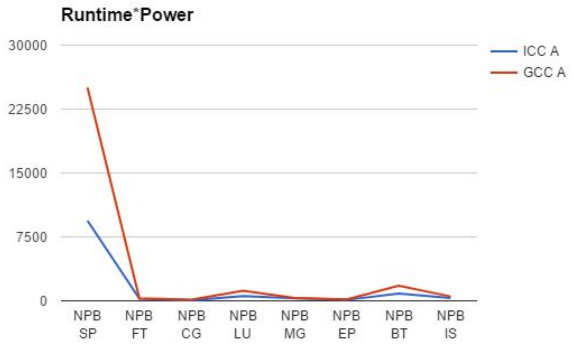

Fig. 11. Runtime*Power comparison GCC vs ICC Class A Dataset

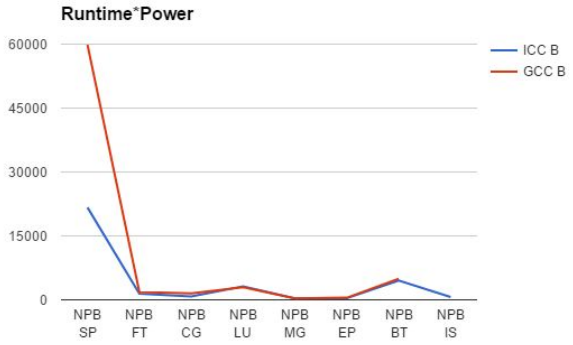

Fig. 12. Runtime*Power comparison GCC vs ICC Class B Dataset

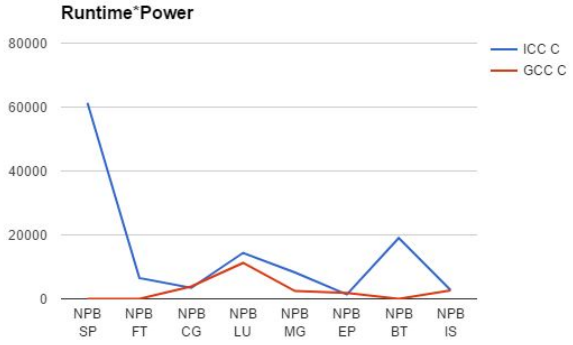

Fig. 13. Runtime*Power comparison GCC vs ICC Class C Dataset

VI. Results

In Experiment I, we found that GCC performance is better than both icc command options we used. In Second experiment, we ICC output-performed GCC in most cases. Still, for Class C datasets we can see GCC outperforms ICC in performance*runtime parameter for some applications. These experiments show the necessity to use latest compilers and command options necessary for optimal compilation for targeted architecture.

The average temperature of icc was 37.06875 and the average temperature of gcc was 36.258955. The clock ran at 1496 Mhz for all experiments. the average runtime for icc was 50.56166667 and average runtime for gcc was 36.258955.

VII. Conclusion and Future Work

We found that using incorrect parameters/ missing compiler parameters using both compilers gives significantly weak overall performance. Using latest compilers gave us better results. ICC outperforms GCC in 6.3.0 version for most applications and most parameters. For large resource consuming applications, having better suited environment including compilers can give significant performance boost. Architecture specific flags such as AVX512 are critical for using new architecture like Intel's x200 KNL. We can also conclude that different applications can be compiled with different compilers and flags given their resource characteristics for best possible results. This can be possible future work from this study.

VIII. Appendix

The project repository can be found on github in this link:
https://github.com/dhruv-jain/energy_efficient_computing

Acknowledgment

We would like to thank Dr. Ge for KNL system access and guidance in the project. We would like to thank Intel for providing VTUNE Amplifier for systems trial version for one of our experiments leading up to this project.

References

[1] https://www.greentechmedia.com/articles/read/improving-the-energy-efficiency-of-computing-while-moores-law-slows
[2] https://en.wikipedia.org/wiki/Xeon_Phi#Knights_Landing.
[3] https://askubuntu.com/questions/464226/how-to-get-cpu-usage-in-percentages
[4] https://software.intel.com/en-us/articles/compiling-for-the-intel-xeon-phi-processor-and-the-intel-avx-512-isa
[5] https://github.com/RRZE-HPC/likwid
[6] https://www.nas.nasa.gov/publications/npb.html
[7] https://github.com/dhruv-jain/energy_efficient_computing