Performance improvement options of scientific applications on XeonPhi KNL architectures

Shajulin Benedict

Indian Institute of Information Technology Kottayam,
Kerala, India
Email: shajulin@iiitkottayam.ac.in

Abstract: Intel's recent manycore processor KNights Landing (KNL) promises high performance for scientific applications. Careful tuning for the complex chip architecture is required to efficiently exploit the chip's hardware resources. This paper describes performance improvement techniques and demonstrates their effectiveness for scientific applications. Experiments were conducted with some of the National Aeronautics and Space Administration (NASA's) advanced supercomputing (NAS) parallel benchmarks, and the effectiveness of: 1) advanced vector extensions (AVX-512) vectorisation support; 2) manycore threading support; 3) the utilisation of thread affinities for different KNL modes, was analysed.

Keywords: knights landing; performance analysis; performance tuning; scientific applications.

Reference to this paper should be made as follows: Benedict, S. (2018) 'Performance improvement options of scientific applications on XeonPhi KNL architectures', *Int. J. Knowledge Engineering and Data Mining*, Vol. 5, Nos. 1/2, pp.1–16.

Biographical notes: Shajulin Benedict graduated from the Manonmaniam Sunderanar University with distinction in 2001. In 2004, he received his ME in Digital Communication and Computer Networking from the A.K.C.E, Anna University-Chennai. He obtained his PhD at the same university. After his PhD award, he joined a research team in Germany to pursue his post doctorate under the guidance of Prof. Gerndt. Later, he worked as a Guest Professor of the TUM, Germany. He is currently working at the Indian Institute of Information Technology Kottayam, Kerala, India, an institute of national importance under MHRD-(PPP act). His research interests include grid/cloud scheduling, performance analysis of parallel applications, IoT and so forth.

1 Introduction

High performance computing (HPC) constantly evolves based on novel computing architectures, scalable algorithms (Müller et al., 2018), and energy efficient solutions (O'Brien et al., 2017; Kaushik and Vidyarthi, 2018; Xiong et al., 2017; Digalwar et al., 2017) for solving science problems, year by year. These updates have

consistently retained HPC researchers over decades for solving the emerging challenges and fine-tuning the available solutions at various levels of implementing scientific applications.

In the past, supercomputer machines had only CPU cores rather than employing hardware accelerators. In order to improve the speed of certain algorithms, mostly data parallel algorithms, supercomputers had included heterogeneous architectures – i.e., the general purpose CPUs were assisted with hardware accelerators. In fact, such heterogeneous architectures have reached top ranks in the Top500 list of supercomputing machines (Top500, 2018). Accordingly performance improvement (PI) options for accelerator-based applications evolved (Zhang et al., 2018; Jararweh et al., 2017).

In recent years, Intel's KNights Landing (KNL) architecture (Sodani, 2015) has attracted several HPC application developers (Adams et al., 2017), especially data scientists working on machine learning or deep learning or 3D visualisation concepts, owing to the possibility of providing over peta-scale performance using KNL CPU cores – KNL achieves the accelerator performance with its bootable processor feature.

Additionally, KNL has a scalable system framework which could be reconfigured to obtain varying compute units, integrated memory, and fabrics. However, it requires a diligent tuning of applications in order to accomplish the maximum performance. This includes, enabling vectorisation support, optimising memory access patterns, pinning required number of threads/processors (based on data locality), and so forth, for scientific applications.

The contributions of the paper are listed as follows:

1 It explores single core tuning techniques.

2 It investigates the tuning techniques for utilising the many cores of the KNL.

3 Iit manifests the performance efficiency aspects of KNL architecture due to choosing appropriate KNL memory modes.

Experiments were conducted by executing the hybrid and openMP versions of NAS parallel benchmarks (NPB) at the KNL partition of Tarus, Technical University – Dresden, Germany. NPB are, in general, widely utilised by several researchers for analysing the performance of applications and architectures (Rosales et al., 2016; Hassan et al., 2016; Narayana et al., 2017).

The remaining sections of the paper contain the following: Section 2 discusses previous works; Section 3 describes the KNL architecture; Section 4 describes the supportive performance efficiency features of applications on KNL; Section 5 illustrates the findings towards vectorisation, manycore features, and data locality options for the NPB and Section 6 presents a few outlooks and conclusions.

2 Related work

Performance analysis of HPC applications (Mishra and Mishra, 2017; Li et al., 2017; Beserra et al., 2017; Gajić et al., 2017; Gong et al., 2018; Benedict et al., 2017) and tool assisted performance monitoring of applications are remaining as a mandatory step to HPC or cloud application developments for decades. Tools such as paradyn (Miller et al., 1995; Roth and Miller, 2006), periscope tool (Benedict and Gerndt, 2012), scalasca

toolkit (Geimer et al., 2006), tuning and utility (TAU) (Shende and Malony, 2005), vampir (Knufer et al., 2008), mpiP (Jerey and Chambreau, 2018) and so forth had emerged for analysing the OpenMP or message passing interface (MPI)-based scientific applications on various architectures.

As newer architectures and applications are evolving, there were efforts to tune the performance of applications with minimal user efforts. The performance tuning efforts were based on performance metrics such as execution time, number of threads, energy consumption, utilisation factor, and so forth; the tuning processes were intended to be a manual or an automatic process. A few researchers have studied the ways to tune linear algebra kernels for improving the energy efficiency of them (Jakobs et al., 2018), a few researchers have framed an energy tuning framework that automatically exploits the dynamism of applications (Schuchart et al., 2017). **?**l and Wasi-ur-Rahman et al. (2017) have tuned mapreduce-based applications with a few tuning parameters.

In the recent past, Intel's many integrated core (MIC) architectures have attracted industrial and academic researchers for tuning applications to accomplish high performance. Intel has motivated researchers by releasing a catalog of applications for MIC machines (Intel, 2016). Accordingly, a few researchers (Prace Info., 2017) have analysed the potential of utilising MIC for their applications. For instances, FeastFlow was analysed by Venetis et al. (2016), scalability and data parallelism patterns were studied for stencil applications (Cebrián et al., 2017).

Memory access patterns and the hierarchical aspects of KNLs memories were discussed by a few researchers in the past. For instances, Perarnau et al. (2016) have discussed the PI achieved for applications while migrating data between the different memories of the KNL. Doerfler et al. (2016) have provided a theoretical analysis of memory bandwidth and memory hierarchies using visual roofline performance models.

The scalability of application on KNL or KNights Corner (KNC) was studied by a few researchers. For instances, Dykes et al. (2016) have studied the scalability feature of KNL machines while experimenting a Splotch visualisation algorithm. Their comparative study of MIC and graphical processing units (GPUs) has shown that hybrid programming of applications could improve the performance of Intel's KNC.

Zhang et al. (2017) have endeavored to study the efficiency of executing deep learning models on multi-node KNL machines. Their early results, as shown in their paper, achieved around 80 percentage efficiency in scalability results for caffe application. Haidar et al. (2016) have studied the scalability aspects of algorithms such as lower upper (LU), QR, and cholesky factorisations. They proposed a programming model to efficiently utilise many core machines such as KNL.

Recently, with the emergence of Intel's KNL architecture, HPC researchers have endeavored to fine tune their applications or do a performance analysis study of applications in order to ease parallelism on a KNL machine. For instances, Barnes et al. (2016) have analysed National Energy Research Scientific Computing Cente (NERSC) workloads on KNL; Afanasyev and Voevodin (2017) and Liu et al. (2017) have manifested that executing graph algorithms in KNL is better than GPUs – in these works, the authors have investigated the cases where manycore features and scalability features of applications could be improved in KNL; Hirokawa et al. (2018) have studied the PI options while porting and executing their Ab-initio real-time electron dynamics (ARTED) code on KNL machines.

This paper has studied the Intel KNL architecture and the PI techniques for HPC applications on KNL.

3 Intel KNL architecture – an overview

This section describes the overview of Intel's KNL architecture and its single node configuration settings in detail.

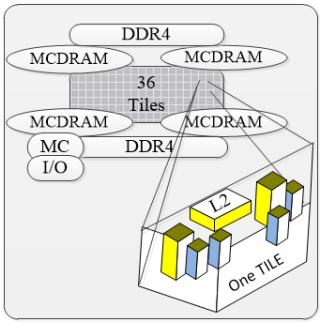
3.1 KNL architecture

Fundamentally, a KNL processor is an extension of a silvermont atom-based microarchitecture which belongs to the MIC series of servers/workstations. The processor units, memory units, IO/memory controllers, and the interconnection units of a KNL machine, are described as below.

3.1.1 Processor units

KNL processor is a bootable manycore architecture with over 8 billion transistors in it. The smallest replicated structure of the KNL processor is called a tile. In KNL, each socket has 36 tiles each tile comprises of two cores and each core embodies two hardware threads and two hyperthreads (see Figure 1). In addition, the processor owns two additional tiles for recovery or for industrial maintenance purposes. The processor is not dependent on serial computer expansion bus standards such as peripheral component interconnect (PCIe), peripheral component interconnect-extended (PCI-X), and so forth; it operates at 1.3 to 1.5 GHz. This architecture, therefore, ease scientific applications to utilise over 256 threads. A pictorial representation of the architecture is shown in Figure 1.

Figure 1 KNL architecture – an overview (see online version for colours)



3.1.2 KNL memory units

KNL includes several levels of memory hierarchies such as,

1 High bandwidth L1 and L2 caches – i.e., L1 cache resides inside a core and L2 cache is shared within the cores of a tile.

2 Two double data rate (DDR4) memory units and

3 Four units of a 16 GB multi-channel dynamic random access memory (MCDRAM). The MCDRAM could be configured in different ways such that it

could feed processors with high bandwidth data of over 450 GB/s (see Section 3.2).

3.1.3 Interconnections

A KNL processor is designed such that a tile is interconnected with the memory/IO controllers (MC/IO), the distributed tag directories (DTD) of tiles, and the other tiles of the node using a mesh ring topology. This interconnection is laid in the IC (die). Hence, KNL is also named as an *on-die mesh interconnect* architecture. This interconnect could be configured to achieve the variable high performance interconnects over 700 GB/s.

3.2 KNL single node configurations

Numbers of configurations and varieties of performance options could be laid out in KNL, which cater the needs of scientific application developers. The single node configurations of KNL are classified in this paper as memory mode and cluster modes of KNL (see Figure 2).

3.2.1 Memory mode configurations

The memory units of KNL could be configured in three different modes namely, flat mode, cache mode, and hybrid mode. These memory modes are initiated in the basic input/output system (BIOS) settings of a machine. Rebooting KNL for changing the memory modes is, therefore, necessary as the appropriate circuits of the KNL circuit board have to be powered on.

The characteristics of these memory modes of KNL are discussed as below:

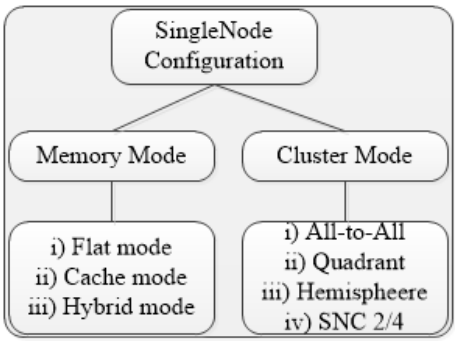
1 *Flat mode:* If KNL is configured as a flat memory mode machine, it operates the available blocks of MCDRAM units as an addressable memory unit of a machine.

2 *Cache mode:* In this mode of operation, MCDRAM could be utilised as cache memory when applications were executed. Hence, applications that require high bandwidth processing may suit well with this mode. However, in worst case scenarios, especially when cache misses occur, the performance of applications might not perform well.

3 *Hybrid mode:* Profiting exclusively by the flat mode or the cache mode might not be a candidate solution for many clustered applications. In hybrid mode, a few portion of the memory units are configured as flat and the other portion as cache. For instance, if an MPI application needs to be executed on a cluster, then hybrid memory mode of KNL could be a performance efficient choice.

Only a few applications will be successful in this mode of operation because the performance of the application might suffer if the data is not available in the nearest MCDRAM block of KNL.

Figure 2 KNL – single node configurations



3.2.2 Cluster mode configurations

In general, application data, including the instructions required for processing an application, are accessed in caches. If they are not found in the cache, the CPU controller requests the memory addresses of an application from the next level of memory in a machine. Each tiles of KNL maintains a distributed tag directory (DTD) to achieve cache coherency. Thus, if the DTD of a tile (often referred to as a local cache) finds a cache miss, it reports the miss to all available DTDs. Subsequently, the missing memory address will be found from any one of the memory units. Although KNL maintains the cache coherency, it offers a few clustering options so that the memory addresses of an application are invoked with limited latency.

The advantages and the characteristics of these clustering modes are described as follows:

1 *All-to-all cluster:* During this clustering mode of KNL, a tile queries all DTDs and searches for the required memory addresses from any available tiles during the cache misses. In this approach, the application might experience hefty latencies while finding the appropriate cache line. Hence, *all-to-all cluster* mode is not often set as a default mode for executing applications.

2 *Quadrant mode:* In this mode, the tiles of a KNL node are equally divided into four equal quadrants. In doing so, the missing cache lines are initially queried with the DTDs belonging to the respective quadrants. Thus, the latency could be lowered when compared to the previous mode of operation.

3 *Hemisphere mode:* This mode is similar to the quadrant clustering mode. However, instead of having four logical divisions, this mode has only two quadrants. In general, quadrant or hemisphere modes are set as default mode for executing applications, especially when MPI/openMP hybrid applications were considered.

4 *Sub-NUMA cluster (SNC) mode:* SNC mode provides additional flexibility to the user. In addition to providing a quadrant space, KNL operates the corresponding MCDRAM memory unit of a quadrant in a non-uniform memory access (NUMA) fashion. This mode is well suited for MPI applications that require more MPI ranks per processor.

3.3 Advantages of KNL

Scientific application developers and the users could be benefited with this architecture in various ways as listed below:

1 The most important advantage of KNL is that it has the capability of working as a standalone booting machine. In contrary, its previous architecture, KNC, requires a host processor to offload instructions to accelerators for imparting parallelism. Hence, KNC had performance concerns while transferring data from the host processor to co-processors;

2 Due to the availability of over 256 working threads in a socket, the existing scientific applications could be scaled well. In addition, several parallel algorithms or energy efficient scalable applications could be developed based on this architecture; and,

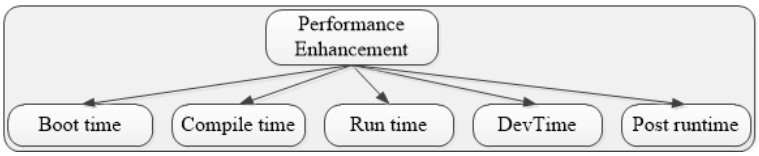
3 An application need not be recompiled for MIC standards. Hence, any scientific applications that are developed for a generic computing architecture could be experimented with KNL.

4 PI techniques

This section discusses on the PI techniques of scientific applications while executing them on KNL architectures.

KNL is designed such that the performance of an application could be tweaked at various stages of its execution, including the boot time of a machine. In this subsection, a few PI options of KNL are categorised and discussed (see Figure 3).

Figure 3 KNL performance enhancement options



4.1 Boot time oriented PI approach

As KNL offers a few pre-defined BIOS configurations such as, selecting the clustering and memory modes of KNL at its boottime, the machine could be configured understanding the characteristics of applications. For instances, a few scalability aware parallel applications require sufficient caches in order to avoid cache contention; and, a few other applications that have irregular data access patterns require flat memory mode.

4.2 Compile time oriented PI approach – vectorisation

Vectorising a few code portions of an application is an important aspect of KNL. The performance of a few applications, in general, could be immensely improved while

enabling vectorisation flags such as -xMIC-AVX512 or xMIC-AVX2, or so forth. The acronymn AVX stands for advanced vector extensions. For instances, the performance of the gather and scatter instructions of scientific applications (which have irregular data access patterns) could be improved when the KNL specific (hardware pre-fetching) compile time flags (-xMIC-AVX512-PF) were enabled. Similarly, aligning data at compile time avoids unnecessary cache misses in KNL. Thus, in succinct, choosing appropriate compiler optimisation switches improves the performance of applications.

4.3 Run time oriented PI approach

In addition to choosing a few compiler optimisation flags, the performance of KNL-based applications could be improved by tweaking applications at runtime. This includes mechanisms such as, imparting scalability and energy aware solutions – selecting the right number of threads and processes at runtime could improve the performance of an application. The users of KNL could apply a few environmental variables for enhancing the performance of applications. For instances, *numactl* linux commands could be utilised for binding applications to a specific NUMA node of KNL and KMP_AFFINITY variable could be set for providing the data locality of applications.

Another approach is to utilise online performance analysis tools for identifying and manually rectifying the performance concerns of applications (Benedict et al., 2015; Benedict and Gerndt, 2012; Geimer et al., 2012). There exist several online performance analysis tools such as, periscope, scalasca, TAU, and so forth. And, the energy specific analysis of applications could be accomplished using a few tools such as, powertop, score-E, energy analyser (Benedict et al., 2015) and so forth (Hahnel et al., 2015).

Apart from normal monitoring and analysis, *autotuning* tools could be applied for automatically tuning a few portions of applications with respect to the underlying machines. To note, KNL oriented autotuning tools are either not available or they are not in the production stage.

4.4 Program development time PI approach

Utilising an appropriate programming language for writing scientific applications could enhance the performance of applications. Since KNL has over 256 working threads (including different clustering modes), OpenMP or buffer-based MPI programming models or MPI+OpenMP programming models could be utilised for writing applications.

Scientific applications could be modified to include SIMD intrinsic at higher level programming in order to support AVX512 instructions. These intrinsic are, in general, written in programs to enable compilers to vectorise the code. The impact of utilising such intrinsic in programs could be analysed by checking the assembler code of the program. Notably, there exists a few assembly level programmers to improve the performance of applications, including vectorisation.

4.5 Post runtime oriented PI approach

A few researchers analyse the performance of scientific applications after executing them. In fact, there exist tools that analyse the performance or vectorisation efficiency of

applications. For instances, *Intel advisor* tool from Intel inc. has a provision to undergo *survey analysis* and *trip count analysis* of applications. To do so, the application has to be compiled with the MIC-based compiler flag (-xMIC-AVX512) prior to its execution. Once after executing the application, the trace report will be generated which could be further analysed for performance problems.

The survey report analysis is utilised to understand the bottlenecks of applications; it pinpoints to any inefficient portions of the vectored code; it suggests for vectoring a few portions of code; and, so forth. In trip count analysis, the tool specifies the number of occurrences of loops. Scientific applications could be traced and the performance insights of these applications could be verified by a few performance analysis tools with robust visualisation of parallel runtime.

5 Experimental results

The previous section classified a number of PI techniques into boot time, compile time, and run time techniques. This section presents the results obtained for a number of NPB and the experimental setup.

5.1 Experimental setup

All experiments, discussed in this paper, were executed at the KNL nodes available at the Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH) – Technical University Dresden. For conducting experiments, the KNL nodes were remotely accessed and the applications were executed on the KNL node by submitting the jobs using the slurm batch system of the Taurus machine. In order to pursue experiments with varying KNL configurations (which should be modified at the BIOS mode of the machine), the nodes were rebooted with specific configurations and jobs were scheduled to these nodes based on Slurm reservations.

In the experiments, NAS benchmark version 3.3.1 MultiZone (MZ) (Van der Wijngaart and Jin, 2003) suite was utilised which is widely utilised by HPC researchers in various contexts in order to understand the parallel performance of machines or algorithms (Sundriya and Sosonkina, 2018). In addition, the NAS benchmarks were designed such that the number of processors, problem size, memory usage, and so forth can be defined prior to their execution. Throughout the experiments, the class C problem size (16 x 16) of NAS was compiled using the *intel* compiler with the *Intel MPI* library.

In succinct, the NAS-version-3.3.1-MZ includes three hybrid benchmarks named as block tri-diagonal (BT) solver, scalar pentadiagonal solver (SP) and LU Gauss siedel solver. These benchmarks are written in Fortran77 with MPI+OpenMP. These three pseudo applications solve partial differential equations (PDE) resulting from the Naiver stokes equation. During the execution, the rectangular problem size is discretised into tiles of 3D meshes. Although these applications attempt to solve the same PDE, the utilisation of networking zones vary among BT, SP and LU. For instances, the BT application has increasing number of networking zones with varying problem sizes – i.e., BT-class A problem size has x = 4, y = 4, z = 1 zones while BT-class C has x = 16, y = 16, z = 1 zones and BT-class D has x = 32, y = 32 and z = 1 zones, SP and

LU has fixed number of networking zones with increasing problem sizes – i.e., classes A, B, C, D and E of SB and LU have x = 4, y = 4 and z = 1 networking zones.

5.2 Boot time oriented PI options

The discussion in this subsection was driven in two parts:

1 At first, the applications when experimented with the SNC-cache mode of KNL was illustrated. In addition, the impact of performance of applications while applying thread affinities namely, compact, scatter, and balanced, was analysed.

2 Later, three different combinations of clustering and memory modes of KNL (SNC-cache, quadrant-cache, and SNC-flat) were compared for these applications.

5.2.1 SNC-cache mode

Here, the NAS hybrid applications (MPI+OpenMP) were executed on the KNL machine using the SNC with cache mode. The NAS applications were run with four MPI ranks and 36 threads per rank, totaling to 144 threads. In the experiments, a few environmental variables, as shown below, were utilised to initiate the defined affinities. For instances,

```
export KMP_AFFINITY=scatter,verbose
export KMP_AFFINITY=compact,verbose
export KMP_AFFINITY=balanced,verbose
```

While testing them with the SNC-cache mode of KNL, OpenMP threads were assigned to the KNL tiles in three different thread affinity policies:

1 *Compact:* Here, each core is filled with the maximum possible number of threads before assigning threads to the another core. This could have poor load balancing as few cores might not have a sufficient number of threads.

2 *Scatter:* In scatter, the threads are distributed among the available cores one at a time so that all cores are evenly loaded. However, a few cores might remain underutilised in a few cases.

3 *Balanced:* In this affinity case, OpenMP threads are evenly load balanced while executing an application.

Figure 4 shows that scatter and balanced affinity policies and SNC-cache mode result in better execution time compared to the compact policy.

5.2.2 Comparison of SNC-cache, quadrant-cache and SNC-flat

With the same number of threads and MPI ranks, the applications were executed in order to understand the impact of three KNL modes, namely, SNC-cache, quadrant-cache and SNC-flat modes. For instance, SNC-cache means that the KNL node was configured with SNC-4 and cache memory modes. The number 4 in SNC-4 resembles the number of blocks of tiles in a KNL. The modes of KNL were explained in the previous

section. Similar explanation follows to the other modes of KNL. In these experiments, applications were fixed to the *scatter* thread affinity option.

Figure 4 SNC-cache mode of KNL – comparison of OpenMP thread affinities policies (see online version for colours)

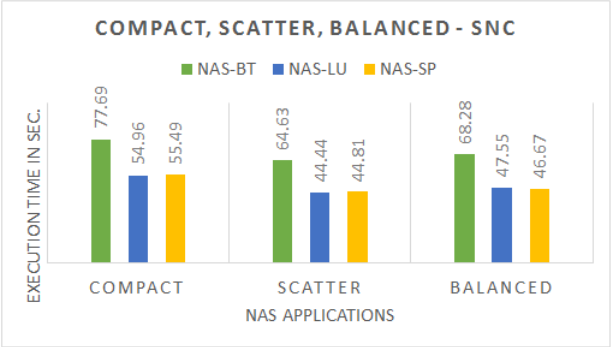


Table 1 shows the execution times of applications in seconds when executed at three memory modes of KNL.

Table 1 Comparison of SNC-cache, quadrant-cache and SNC-flat

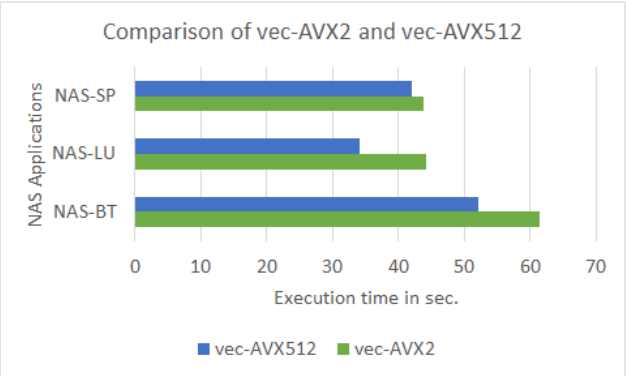
NAS	*SNC-cache*	*Quadrant-cache*	*SNC-flat*
NAS-BT	77.69	62.48	493.89
NAS-LU	54.96	44.86	435.53
NAS-SP	55.49	43.98	299.12

It can be noticed from this table that the SNC-flat mode of KNL had worst performances when compared to the other modes. This was due to the fact that the NAS applications had to refer to the MCDRAM memory units as addressable memories in the case of SNC-F mode. Whereas, the first two columns of the table, referring to SNC-cache and quadrant-cache modes, had utilised the high bandwidth capability of KNL (as MCDRAM acted as cache).

5.3 Impact of vectorisation support – vec2 vs. vec512 (compile time oriented)

In general, vectorisation is effective for data intensive applications that have non-dependent loops. Here, vectorisation was applied to the NAS applications using compiler flags. These flags enabled the generation of AVX2 instructions and AVX512 instructions based on the flags (-xCORE-AVX2 and -xMIC-AVX512). Figure 5 vividly pinpoints that the applications vectorised for AVX512 (KNL-specific) surpassed AVX2 for all cases. Notably, LU and BT had a hefty PI when comcompilled forfor the AVX512 instructions.

Figure 5 Comparison of utilising vector instructions AVX2 and AVX512 (see online version for colours)



5.4 *Effects on number of threads (runtime oriented)*

It was also interesting to understand the efficiency of KNL in terms of number of threads. To illustrate this scenario, in this experiment, the OpenMP version of NAS parallel benchmark-BT class A was run. In addition, experiments were conducted considering different thread affinities while increasing the number of threads from 1 to 256.

Table 2 shows the execution time of NAS-BT. As seen in the table, the execution time improved with a increasing number of threads. However, the performance started to deteriorate from 128 threads. These findings got reflected in the three affinity cases of the performance study of NAS-BT.

Table 2 NAS-BT on KNL – impact of varying number of threads

No. of threads	*Execution time in sec.*		
	Compact	*Scatter*	*Balanced*
1	181.18	181.18	181.18
2	108.45	69.2	69.4
4	98.09	36.69	36.78
8	54.79	18.35	18.41
16	27.9	9.26	9.28
32	13.98	4.26	4.69
64	7.06	2.46	2.49
128	6.95	3.86	3.86
256	7.04	6.19	6.21

6 Conclusions

Many-core architectures are promising architectures for scientific application developers belonging to the different scientific communities, including the machine learning-based AI community. Although a few simulation-based research efforts have been carried

out in the past, the performance analysis study of scientific applications in the real test environment is limited. This paper explored and categorised the PI options of scientific applications on KNL machines. Experiments were conducted for NAS hybrid pseudo-applications at the KNL machine of ZIH-TU-Dresden and the performance results from the experiments were discussed.

Acknowledgements

The author thanks the editor of IJKEDM, Prof. Dr. Michael Gerndt, IIITKottayam officials, ZIH-TU-Dresden, Dr. Carsten Trinitis and Intel-Germany/India for supporting and motivating the work.

References

Adams, M.F., Hirvijoki, E., Knepley, M.G., Brown, J., Isaac, T. and Mills, R. (2017) 'Landau collision integral solver with adaptive mesh refinement on emerging architectures', in *SIAM J. Sci. Comput.*, Vol. 39, No. 6, pp.452–465.

Afanasyev, I. and Voevodin, V. (2017) 'The comparison of large-scale graph processing algorithms implementation methods for Intel KNL and NVIDIA GPU', in *Russian Supercomputing Days, Supercomputing, RuSCDays 2017, Communications in Computer and Information Science*, Vol. 793, pp.80–94, Springer.

Barnes, T., Cook, B., Deslippe, J., Doerfler, D., Friesen, B., He, Y., Kurth, T., Koskela, T., Lobet, M., Malas, T., Oliker, L., Ovsyannikov, A., Sarje, A., Vay, J-L., Vincenti, H., Williams, S., Carrier, P., Wichmann, N., Wagner, M., Kent, P., Kerr, C. and Dennis, J. (2016) 'Evaluating and optimizing the NERSC workload on knights landing', in *Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PMBS '16)*, IEEE Press, Piscataway, NJ, USA, pp.43–53.

Benedict, S. and Gerndt, M. (2012) 'Automatic performance analysis of OpenMP codes on a scalable shared memory system using periscope', *Applied Parallel and Scientific Computing*, LNCS, Vol. 7134, No. 1, pp.452–462, DOI: 10.1007/978-3-642-28145-7_44.

Benedict, S., Rejitha, R.S. and Bright C. (2015) 'Energy consumption analysis of HPC applications using NoSQL database feature of energy analyzer', in *LNCS of ICC2014, Intelligent Cloud Computing*, Springer, pp.103–118.

Benedict, S., Rejitha, R.S., Preethi, C., Bright, C.B. and Judyfer, W.S. (2017) 'Energy analysis of code regions of HPC applications using energy analyzer tool', in *International Journal of Computational Science and Engineering*, Vol. 14, No. 3, pp.267–278.

Beserra, D., Moreno, E.D., Endo, P.T., Barreto, J., Fernandes, S. and Sadok, D. (2017) 'Performance analysis of Linux containers for high performance computing applications', in *International Journal of Grid and Utility Computing*, Vol. 8, No. 4, pp.321–329.

Cebrián, J.M., Cecilia, J.M., Hernández, M. and García, J.M. (2017) 'Code modernization strategies to 3-D stencil-based applications on Intel Xeon Phi: KNC and KNL', *Computers & Mathematics with Applications*, Vol. 74, No. 10, pp.2557–2571.

Cheng, D., Rao, J., Guo, Y., Jiang, C. and Zhou, X. (2017) 'Improving performance of heterogeneous map reduce clusters with adaptive task tuning', in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 3, pp.774–786.

Digalwar, M., Gahukar, P., Raveendran, B.K. and Mohan, S. (2017) 'Energy efficient real-time scheduling algorithm for mixed task set on multi-core processors', *International Journal of Embedded Systems*, Vol. 9, No. 6, pp.523–534.

Doerfler, D., Deslippe, J., Williams, S., Oliker, L., Cook, B., Kurth, T., Lobet, M., Malas, T., Vay, J-L. and Vincenti, H. (2016) 'Applying the roofline performance model to the Intel Xeon Phi knights landing processor', in *ISC High Performance, Lecture Notes in Computer Science*, Vol. 9945, pp.339–353.

Dykes, T., Gheller, C., Rivi, M. and Krokos, M. (2016) 'Splotch: porting and optimizing for the Xeon Phi', in *Int. Journal of High Perf. Comp. Applns.*, Vol. 31, No. 6, pp.550–563.

Gajić, D.B., Stanković, R.S. and Radmanović, M. (2017) 'A performance analysis of computing the LU and the QR matrix decompositions on the CPU and the GPU', *Int. J. of Reasoning-based Intelligent Systems*, Vol. 9, No. 2, pp.114–121.

Geimer, M., Wolf, F., Wylie, B.J.N. and Mohr, B. (2006) 'Scalable parallel trace-based performance analysis', in *Proc. of the 13th Eur. PVM/MPI Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2006)*, Bonn, Germany, pp.303–312.

Geimer M., Saviankou, P., Strube, A., Szebenyi, Z., Wolf, F. and Wylie, B.J.N. (2012) 'Further improving the scalability of the scalasca toolset', *LNCS*, Vol. 7134, No. 1, pp.463–474, Springer, DOI: 10.1007/978-3-642-28145-7_45.

Gong, X., Shao, Z., Du, Q., Yu, A., Zhang, J. (2018) 'Performance overhead analysis of virtualisation on ARM', *International Journal of Information and Communication Technology*, Vol. 12, Nos. 1–2, pp.143–161.

Hahnel M., Dobel B., Volp M. and Hartig H. (2015) *Measuring Energy Consumption for Short Code Paths Using RAPL* [online] http://www.sigmetrics.org/greenmetrics/Hahnel.pdf (accessed 13 January 2016).

Haidar A., Tomov, S., Arturov, K., Guney, M., Story, S. and Dongarra, J. (2016) 'LU, QR, and Cholesky factorizations:pProgramming model, performance analysis and optimization techniques for the Intel knights landing Xeon Phi', in *Proc. of IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, pp.1–7.

Hassan, H.A., Mohamed, S.A. and Sheta, W.M. (2016) 'Scalability and communication performance of HPC on Azure cloud', *Egyptian Informatics Journal*, Vol. 17, No. 2, pp.175–182.

Hirokawa, Y., Boku, T., Sato, S.A. and Yabana, K. (2018) 'Performance evaluation of large scale electron dynamics simulation under many-core cluster based on knights landing', in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2018)*, ACM, New York, NY, USA, pp.183–191 [online] https://doi.org/10.1145/3149457.3149465 (accessed 3 April 2018).

Intel (2016) *Intel Xeon Phi Coprocessor Applications and Solutions* [online] https://software.intel.com/en-us/xeonphionlinecatalog (accessed 1 February 2017).

Jakobs, T., Lang, J., Rünger, G. and Stöcker, P. (2018) 'Tuning linear algebra for energy efficiency on multicore machines by adapting the ATLAS library', *Future Generation Computer Systems*, Vol. 82, No. 9, pp.555–564.

Jararweh, Y., Al-Ayyoub, M., Fakirah, M., Alawneh, L. and Gupta, B.B. (2017) 'Improving the performance of the needleman-wunsch algorithm using parallelization and vectorization techniques', in *Multimedia Tools and Applications*, pp.1–17, Springer.

Jerey, V. and Chambreau, C. (2018) *mpiP: Lightweight, Scalable MPI Profiling* [online] http://mpip.sourceforge.net (accessed 12 March 2017).

Kaushik, A. and Vidyarthi, D.P. (2018) 'A hybrid heuristic resource allocation model for computational grid for optimal energy usage', in *International Journal of Grid and Utility Computing*, Vol. 9, No. 1, pp.51–74.

Knupfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Mller, M.S. and Nagel, W.E. (2008) 'The vampir performance analysis toolset', in *Proc. of the 2nd Int. Work. on Parallel Tools for HPC*, HLRS, Stuttgart, pp.139–155, Springer Publications.

Li, H., Xu, X., Tang, Y. and Ren, X. (2017) 'A multi-user performance analysis framework for CFD simulations', in *Int. Journal of Progress in Computational Fluid Dynamics*, Vol. 17, No. 4, pp.199–211.

Liu, X., Chen, L., Firoz, J.S., Qiu, J. and Jiang, L. (2017) *Performance Characterization of Multi-threaded Graph Processing Applications on Intel Many-Integrated-Core Architecture* [online] https://arxiv.org/abs/1708.04701 (accessed March 2018).

Müller, A., Kopera, M.A., Marras, S., Wilcox, L.C., Isaac, T. and Giraldo, F.X. (2018) 'Strong scaling for numerical weather prediction at petascale with the atmospheric model NUMA', *The International Journal of High Performance Computing Applications*, April [online] https://doi.org/10.1177109434201876396 6.

Miller B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K. and Newhall, T. (1995) 'The paradyn parallel performance measurement tool', *IEEE Computer*, Vol. 28, No. 11, pp.37–46.

Mishra, J.P. and Mishra, S.K. (2017) 'Evaluating performance with implementation of virtualised data in the cloud using metaheuristic approach', in *Int. J. of Knowledge Engineering and Data Mining*, Vol. 4, Nos. 3/4, pp.187–203.

Narayana, V.K., Sun, S., Badawy, A-H.A., Sorger, V.J. and El-Ghazawi, T. (2017) 'MorphoNoC: exploring the design space of a configurable hybrid NoC using nanophotonics', *Microprocessors and Microsystems*, Vol. 50, No. 3, pp.113–126.

O'Brien, K., Pietri, I., Reddy, R., Lastovetsky, A. and Sakellariou, R. (2017) 'A survey of power and energy predictive models in HPC systems and applications', *ACM Comput. Surv.*, Vol. 50, No. 3, DOI: https://doi.org/10.1145/3078811.

Perarnau, S., Zounmevo, J.A., Gerofi, B., Iskra, K. and Beckman, P. (2016) 'Exploring data migration for future deep-memory many-core systems', in *Proc. of IEEE International Conference on Cluster Computing (CLUSTER)*, Taipei, pp.289–297.

Prace Info. (2017) *Prace – MIC Research Experiences* [online] http://www.prace-ri.eu/evaluation-intel-mic/ (accessed 3 February 2018).

Rosales, C., Cazes, J., Milfeld, K., Gómez-Iglesias, A., Koesterke, L., Huang, L. and Vienne, J. (2016) 'A comparative study of application performance and scalability on the Intel knights landing processor', *ISC High Performance, Lecture Notes in Computer Science*, Vol. 9945, pp.307–318, Springer.

Roth, P.C. and Miller, B.P. (2006) 'The distributed performance consultant and the sub-graph folding algorithm: on-line automated performance diagnosis on thousands of processes', in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'06)*.

Schuchart, J., Gerndt, M., Kjeldsberg, P.G., Lysaght, M., Horák, D., Říha, L., Gocht, A., Sourouri, M., Kumaraswamy, M., Chowdhury, A., Jahre, M., Diethelm, K., Bouizi, O., Mian, U.S., Kruvzík, J., Sojka, R., Beseda, M., Kannan, V., Bendifallah, Z., Hackenberg, D. and Nagel, W.E. (2017) 'The READEX formalism for automatic tuning for energy efficiency', in *Computing*, Vol. 99, No. 8, pp.727–745.

Shende, S.S. and Malony, A.D. (2005) 'The TAU parallel performance system', in *International Journal of High Performance Computing Applications*, ACTS Collection Special Issue, Vol. 20, No. 2, pp.287–311.

Sodani, A. (2015) 'Knights landing (KNL): 2nd generation Intel Xeon Phi processor', in *Proc. of IEEE Hot Chips 27 Symposium (HCS)*, Cupertino, CA, pp.1–24.

Sundriya, V. and Sosonkina, M. (2018) 'Modeling of the CPU frequency to minimize energy consumption in parallel applications', *Sustainable Computing: Informatics and Systems*, Vol. 17, No. 1, pp.1–8.

Top500 (2018) *Supercomputers List* [online] https://www.top500.org/ (accessed 3 January 2018).

Van der Wijngaart, R.F. and Jin, H. (2003) *NAS Parallel Benchmarks, Multi-Zone Versions*, NAS Technical Report, pp.1–9, NAS-03-010.

Venetis, I.E., Goumas, G., Geveler, M. and Ribbrock, D. (2016) *Porting FEASTFLOW to the Intel Xeon Phi: Lessons Learned* [online] http://www.prace-ri.eu/IMG/pdf/wp139.pdf (accessed 2017).

Wasi-ur-Rahman, M., Islam, N.S., Lu, X., Shankar, D. and Panda, D.K. (2017) 'MR-advisor: a comprehensive tuning, profiling, and prediction tool for map reduce execution frameworks on HPC clusters', *Journal of Parallel and Distributed Computing*, in press [online] https://doi.org/10.1016/j.jpdc.2017.11.004 (accessed 5 March 2018).

Xiong, Y., Chen, Y., Jiang, K. and Tang, Y. (2017) 'An energy-aware task consolidation algorithm for cloud computing data centre', *International Journal of High Performance Computing and Networking*, Vol. 10, Nos. 4–5, pp.352–358.

Zhang, Z., Xu, W., Gaffney, N. and Stanzione, D. (2017) 'Early results of deep learning on the stampede2 supercomputer', in *IXPUGFall 2017*, pp.1–3.

Zhang, T., Dai, H., Wu, C. and Jia, Z. (2018) 'Enhancing the performance of process level redundancy with coprocessors in symmetric multiprocessors', in *International Journal of Computational Science and Engineering*, Vol. 16, No. 1, in press.