Peer-reviewed version available at J. Low Power Electron. Appl. 2018, 8, 18; doi:10.3390/jlpea8020018

Article

Software and DVFS tuning for performance and energy-efficiency on Intel KNL processors

Enrico Calore [1]*, Alessandro Gabbana [1,2], Sebastiano Fabio Schifano[1] and Raffaele Tripiccione [1,]

[1] INFN and Università degli Studi di Ferrara, Ferrara, ITALY
[2] Bergische Universität Wuppertal, Wuppertal, GERMANY
* Correspondence: enrico.calore@unife.it; Tel.: +39-0532-974612

Abstract: Energy consumption of processors and memories is quickly becoming a limiting factor in the deployment of large computing systems. For this reason it is important to understand the energy performance of these processors and to study strategies allowing to use them in the most efficient way. In this work we focus on computing and energy performance of the *Knights Landing* Xeon Phi, the latest Intel many-core architecture processor for HPC applications. We consider the 64-core Xeon Phi 7230, and profile its performance and energy efficiency using both its on-chip MCDRAM and the off-chip DDR4 memory as the main storage for application data. As a benchmark application we use a Lattice Boltzmann code heavily optimized for this architecture, and implemented using several different arrangements of the application data in memory (data-layouts, in short). We also assess the dependence of energy consumption on data-layouts, memory configurations (DDR4 or MCDRAM), and number of threads per core. We finally consider possible trade-offs between computing performance and energy efficiency, tuning the clock frequency of the processor using the Dynamic Voltage and Frequency Scaling (DVFS) technique.

Keywords: Energy; KNL; MCDRAM; Memory; Lattice Boltzmann; HPC; DVFS

1. Introduction

Energy consumption is quickly becoming one of the most critical issues in modern HPC systems. Correspondingly, a lot of interest is now focused on attempts to increase energy efficiency using a variety of different, hardware and software-based, approaches. A further driver of researches towards energy efficiency is also given by the need to lower the total cost of ownership of HPC installations, increasingly depending on the electricity bill. Processors and accelerators are the main sources of power drain in modern computing systems [1]; for this reason, assessing their energy-efficiency is of paramount importance for the design and deployment of large energy-efficient parallel systems. Recent hardware developments show a definite trend to improve energy-efficiency, as measured typically by the increasing peak-FLOPs/Watt ratio of recent architectures [2]. However, measuring and profiling the energy performance of actual applications is obviously relevant, as very energy-efficient processors (accordingly to the peak-FLOPs/Watt ratio) may be highly inefficient when running codes unable to exploit a large fraction of their peak performance. For this reason in this work we study the energy-efficiency of the Intel Knights Landing (KNL) architecture, using as a benchmarking code a real HPC application, that has been heavily optimized for several architectures and routinely used for production runs of fluid-dynamics simulations based on the Lattice Boltzmann method [3]. This application is a good representative of a wider class of lattice based stencil-codes, including also HPC *Grand Challenge* applications such as Lattice Quantum Chromodynamics (LQCD) [4–8].

In this work we evaluate the impact of a variety of software options – different number of threads per core, different memory configurations (i.e. MCDRAM and DDR4) and different data layouts – on energy consumption of the KNL as it runs different kernels of our benchmark application. In this paper, which is an extended version of [9], we also explore further trade-offs between performance and energy-efficiency made possible by processor frequency tuning, and assess the corresponding impact on the time-to-solution of our application. To this effect, we introduce a new section discussing the clock-frequency tuning of the KNL using DVFS (Dynamic Voltage and Frequency Scaling) technique. Here we measure the time-to-solution and the energy-to-solution of our benchmark application for different processor frequencies, following an approach already used also for other architectures [10].

2. Lattice Boltzmann Methods

Lattice Boltzmann methods (LB) are a class of numerical schemes routinely used in many physics and engineering contexts, to study the dynamics of fluid flows in several different regimes. LB methods [11] are based on the synthetic dynamics of *populations* sitting at the sites of a discrete lattice. Over the years, many different LB methods have been developed; while there are significant differences among them in terms of their detailed structure, physical and numerical accuracy and application domains, all LB methods are discrete in position and momentum spaces; they all share a basic algorithmic structure in which each discrete time-step is obtained by first moving populations from lattice-sites to neighbouring lattice-sites (the *propagate* step) , and then recomputing the values of all populations at each site (the *collide* step). Macroscopic physical quantities (e.g., density, velocity, temperature) are computed as appropriate weighted averages of all population values.

In this work we consider a state-of-the-art $D2Q37$ LB model that has been extensively used for large scale simulations of convective turbulence [12] on HPC systems [13]. This model uses 37 populations per lattice site, and reproduces the thermo-hydrodynamical evolution of fluids in two dimensions, enforcing the equation of state of a perfect gas ($p = \rho T$) [14,15].

A Lattice Boltzmann simulation code starts with an initial assignment of the populations values, and then evolves the system in time for as many time-steps as needed, spending most of its compute time in the execution of the *propagate* and *collide* kernels. As already remarked, *propagate* moves populations across lattice sites according to a stencil that depends on the LB model used. This kernel only performs a large number of sparse memory accesses, and for this reason it is strongly memory-bound. On the other hand, the *collide* kernel uses as input the populations gathered by *propagate*, and performs all the mathematical steps associated to the computation of the new population values. This function is strongly compute-bound, making heavy use of the floating-point units of the processor.

Over the years, we have developed several implementations of this LB model, using them for studies of convective turbulence [16], and as benchmarks, profiling the performance of recent HPC hardware architectures based on several commodity CPUs and GPUs [17–21]. In this work we use an implementation initially developed for Intel CPUs [22], and later ported and optimized for the Intel *Knights Corner* (KNC) processor [23,24].

3. The Knights Landing Architecture

The *Knights Landing* (KNL) is the first generation self-bootable processor based on the Many Integrated Cores (MIC) architecture developed by Intel. This processor integrates an array of 64, 68 or 72 cores together with four high speed *Multi-Channel DRAM* (MCDRAM) memory banks, providing an aggregate raw peak bandwidth of more than 450 GB/s [25]. It also integrates 6 DDR4 channels supporting up to 384 GB of memory with a peak raw bandwidth of 115.2 GB/s. Two cores are bonded together into a tile sharing an L2-cache of 1 MB. Tiles are connected by a 2D-mesh of rings and can be clustered in several NUMA configurations. In this work we only use the *Quadrant* cluster configuration where the array of tiles is partitioned in four quadrants, each connected to one MCDRAM controller. This configurations is the one recommended by Intel to run applications using the KNL as a symmetric multi-processor, as it reduces the latency of L2-cache misses, and the 4 blocks of MCDRAM appear as contiguous blocks of memory addresses [26]. Additionally, the MCDRAM can be configured at boot time in *Flat*, *Cache* or *Hybrid* mode. The *Flat* mode configures the MCDRAM as a separate addressable memory, while the *Cache* mode configures the MCDRAM as a last-level cache; the *Hybrid* mode allows to use the MCDRAM partly as addressable memory and partly as last-level cache [27]. In this work we only consider *Flat* and *Cache* configurations.

KNL exploits two levels of parallelism, *task parallelism* built onto the array of cores, *data parallelism* using the AVX 512-bit vector (SIMD) instructions. Each core has two out-of-order vector processing units (VPUs) and supports the execution of up to 4 threads. The KNL has a peak theoretical performance of 6 TFlops in single precision and 3 TFlops in double precision, and the typical thermal design power (TDP) is 215 W, including MCDRAM memories. For more details on this architecture see [28].

4. Measuring the Energy Consumption of the KNL

As other Intel processors – from Sandy Bridge architecture onward – the KNL integrates power meters and a set of *Machine Specific Registers* (MSR) that can be read through the *Running Average Power Limit* (RAPL) interface. In this work we use the *PACKAGE_ENERGY* and *DRAM_ENERGY* counters to monitor respectively the energy consumption of the processor Package (accounting for Cores, Caches and MCDRAM) and of the DRAM memory system.

A popular way to access these counters is through a library named PAPI [29] providing a common API for energy/power readings for different processors, partially hiding architectural details. The use of this technique to read energy consumption data from Intel processors is an established practice [30], validated by several third parties studies [31,32]. On top of the PAPI library we have developed a custom library to manage power/energy data acquisition from hardware registers. This library allows benchmarking codes to directly start and stop measurements, using architecture specific interfaces, such as RAPL for Intel CPUs and the *NVIDIA Management Library* (NVML) for NVIDIA GPUs [10]. Our library also lets benchmarking codes to place markers in the data stream in order to have an accurate time correlation between the running kernels and the acquired power/energy values. This library is available for download as Free Software [1].

We use the *energy-to-solution* (E_s) as our basic metric to measure the energy-efficiency of processors running our application. This is defined as product of *time-to-solution* (T_s) and average power (P_{avg}) drained while computing the workload: $E_s = T_s \times P_{avg}$. To measure P_{avg} we read the RAPL hardware counters thanks to our wrapper library, and then convert readout values to Watt. The Package and DRAM power drains can then be summed to obtain the overall value or analyzed separately.

In the following, we measure this metric for the two most time consuming kernels of our application, *propagate* which is strongly memory-bound, and *collide* which is strongly compute-bound.

5. Energy-optimization of data structures

The LB application described in Sec. 2 has been originally implemented using the *AoS* (*Array of Structure*) data layout, showing a good performance on CPU processors. Later, its data layout has been re-factored, porting the application to GPUs, leading to another implementation based on the *SoA* (*Structure of Array*) data layout, giving better performance on these processors.

More recently, two slightly more complex data layouts have been introduced [33] in the quest of a single data structure to be used for a portable implementation, able to obtain high performance on most available architectures. We have considered two hybrid data structures, mixing *AoS* and *SoA* properties, that we call *CSoA* (*Clustered Structure of Array*) and *CAoSoA* (*Clustered Array of Structure of Array*). *CSoA*is a modified *SoA* scheme that guarantees that vectorized read and writes are always aligned. This is obtained clustering a set of consecutive elements of each population array in clusters of VL elements, where VL is a multiple of the hardware vector length. The *CAoSoA* layout is a further optimization of the former, further improving population data locality, as it keeps at close – and not only aligned – addresses all the populations data needed to process each lattice site. A more detailed description of both layouts can be found in [33].

In this work we test all of the above data layouts on the KNL architecture, measuring the energy consumption of both the Package and DRAM. We also run with various numbers of threads and with different memory configurations: i) allocating the lattice data only on the DRAM using the Flat/Quadrant configuration; ii) allocating the lattice only on the MCDRAM using the Flat/Quadrant configuration; or iii) allocating the lattice only on the DRAM, but using the MCDRAM as a last level cache, using the Cache/Quadrant configuration. The last case is relevant when using very large lattices, which do not fit in the MCDRAM. As we consider different lattice sizes we present all our results normalizing them to one lattice site. Our aim is to analyze and highlight possible differences in performance, average power, and energy-efficiency.

---

[1] https://baltig.infn.it/COKA/PAPI-power-reader

Fig. 1a and Fig. 1b show the main results of our tests, where Power and Energy values account for the sum of Package and DRAM contributions and Energy is normalized per lattice site.

Concerning the *propagate* function, as shown in Fig. 1a, using Flat-mode and allocating the lattice in the MCDRAM memory, the maximum bandwidth using the *AoS* data layout is 138 GB/s, while using *SoA* it can be increased to 314 GB/s (i.e., 2.3×), and then further increased to 433 GB/s (3.1× wrt *AoS*)using the *CSoA* layout. When using the main DRAM, bandwidth drops for all data layouts: 51 GB/s for *AoS*, 56 GB/s for *SoA*, and 81 GB/s for *CSoA*. Finally, when using the Cache-mode and a larger lattice size, that does not fit in the MCDRAM, we measure an almost constant value of 59, 60 and 62 GB/s respectively for *AoS*, *SoA* and *CSoA*. The *CAoSoA* data layout does not improve over the *CSoA* for the *propagate* function, but as we discuss later (see however Fig. 1b), it improves the performance of the *collide* kernel.

Our best E_S figure is obtained using a Flat-MCDRAM configuration and the *CSoA* data layout, giving an energy reduction of $\approx 2.5\times$ wrt the *AoS* data layout.

From the point of view of all the evaluated metrics, using just 64 threads is the best choice, since it gives the best performance plus the lowest Power drain and Energy consumption. This is justified by the fact that the *propagate* function is completely memory-bound and 64 threads are enough to keep the memory controllers busy at all times.

Concerning the *collide* function, similar plots are shown in Fig. 1b, where again the average Power drain and E_S are given as the sum of Package and DRAM contributions and E_S is normalized per lattice site. For the Flat-MCDRAM configuration, performance increases when changing the data layout from *AoS* to *CSoA* and in this case it can be further increased changing to *CAoSoA*. On the other side, *SoA* yields the worst performance, as vectorization can be very poorly exploited in this case, and moreover memory-alignment cannot be granted. When using *CAoSoA* we measure a sustained performance of $\approx$ 1Tflops, corresponding to $\approx$ 37% of the raw peak performance of the KNL. It is very nice to remark that the *CAoSoA* layout also gives the best E_S, $\approx 2\times$ better than *AoS*, both for performance and E_S.

Finally, we remark that, at variance with the behavior of *propagate*, for the *collide* function E_S decreases using more threads per CPU since this kernel is compute-bound.

For both functions we see that the E_S differences, between the various tests performed, are mainly driven by T_S. Despite the fact that the average Power drain shows differences up to $\approx$ 30%, it seems always convenient to choose the best performing configuration from the T_S point of view, to obtain also the best energy-efficiency.

In Fig. 2 we highlight the E_S metrics for all the different data layouts, using the Flat configuration and thus using either the off-chip (DDR4) or the on-chip (MCDRAM) memory, to allocate the whole lattice. Here we display separately the Package and DRAM contributions, where for each bar in the plot, Package energy is at the bottom and DRAM energy at the top. When using the MCDRAM, its energy consumption is accounted with the rest of the Package and thus what is displayed as DRAM energy is just due to the idle Power drain.

The main advantage in using the MCDRAM is clearly for the *propagate* function where we save $\approx 2/3$ of the energy wrt the best performing test run using the DDR4 system memory (be aware of the different scales in the y-axes of Fig. 2a). Anyhow, also for the *collide* function, shown in Fig. 2b we can halve the energy consumption. In both cases the energy saving is mainly given by the reduced execution time, since the DDR4 average Power drain accounts at most for $\approx$ 10% of the total.

6. Energy-efficiency optimization using DVFS

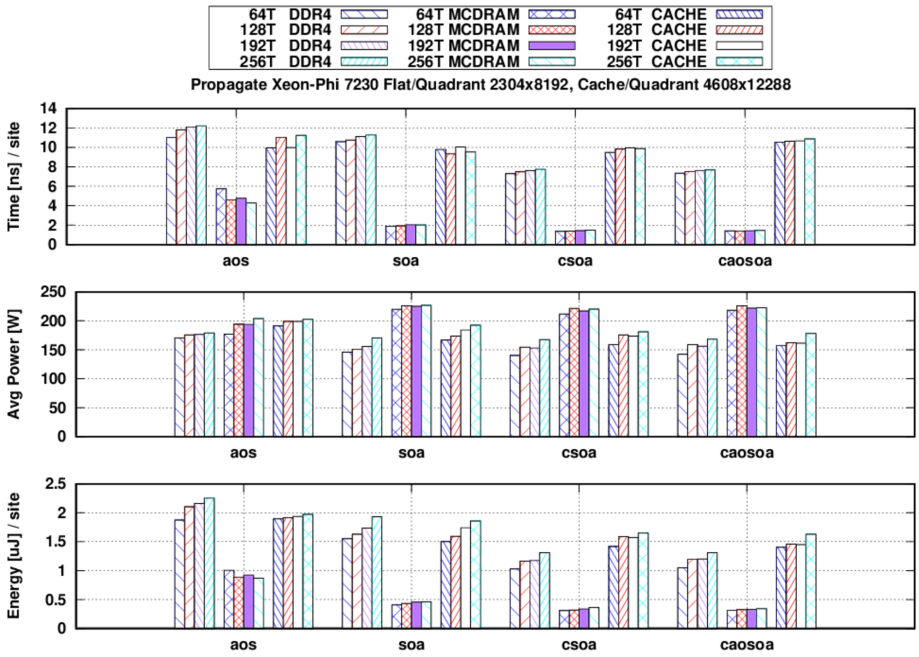
A handy way for users to tune power-related parameters of processors is to use the DVFS support. This is available on most recent processors, giving the possibility to set a specific processor clock frequency [34].

To further investigate possible optimization towards energy-efficiency, we then select the version of our LBM code giving the best computing performance (i.e., the one using the *CAoSoA* data layout), aiming to explore the best trade-off between time-to-solution and energy-to-solution changing the clock frequency of the KNL processor.

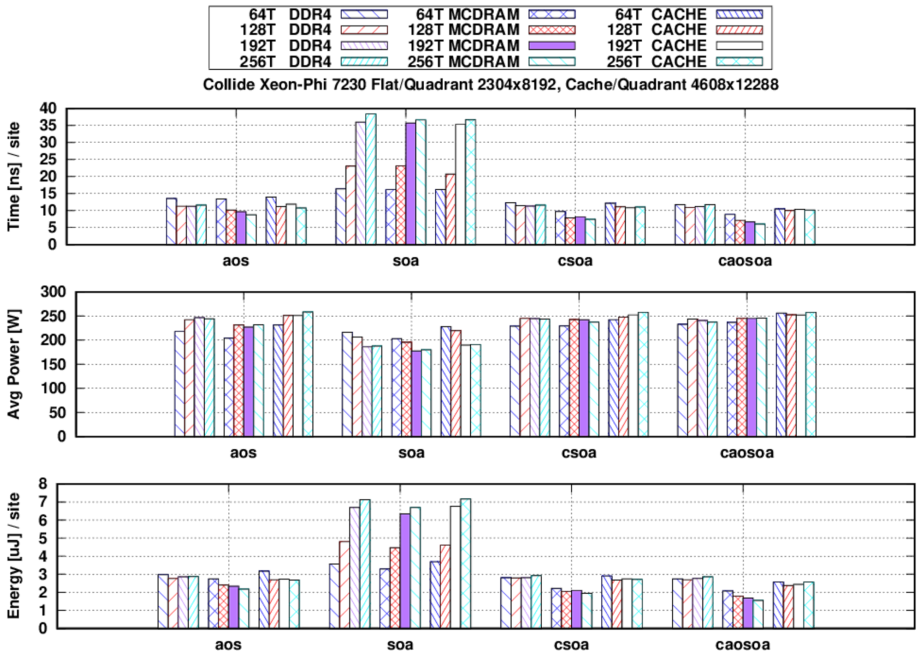
Recently, other research works have focused on the KNL architecture, setting different power caps to force a frequency and voltage change, while running different benchmarks [35,36]. We have also recently used the same LB application, for a similar study on different architectures [10,19], explicitly setting different specific

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 April 2018 doi:10.20944/preprints201804.0074.v1

Peer-reviewed version available at J. Low Power Electron. Appl. 2018, 8, 18; doi:10.3390/jlpea8020018



(a) *propagate* function: T_s in nanoseconds per site (Top), P_{avg} in Watt (Middle) and E_s in microjoules per site (Bottom).

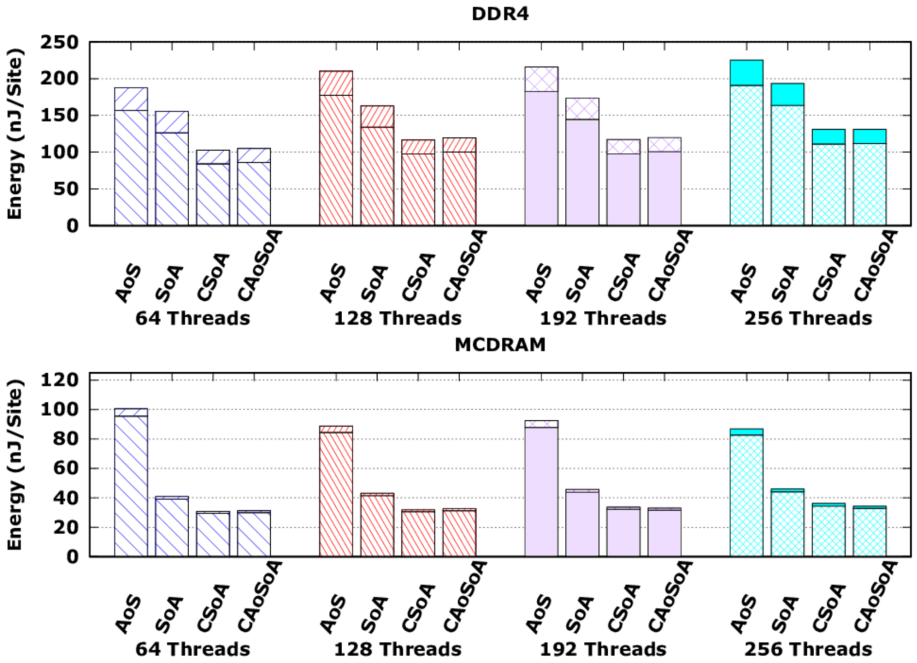


(b) *collide* function: T_s in nanoseconds per site (Top), P_{avg} in Watt (Middle) and E_s in microjoules per site (Bottom).

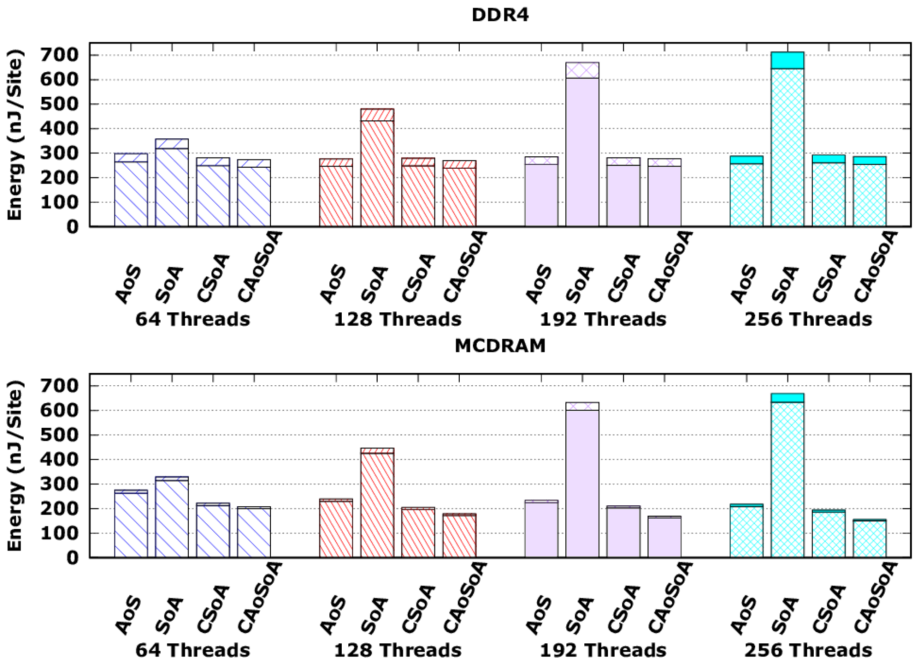
Figure 1. Tests run using three different memory configurations (DDR4 Flat, MCDRAM Flat and Cache) and number of threads (64, 128, 192 and 256). We show three metrics: time-to-solution (T_s), average Power drain (P_{avg}) and energy-to-solution (E_s). Fig. 1a refers to the *propagate* and Fig. 1b to the *collide* kernels.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 April 2018 doi:10.20944/preprints201804.0074.v1

Peer-reviewed version available at J. Low Power Electron. Appl. 2018, 8, 18; doi:10.3390/jlpea8020018



(a) *propagate* function. Flat configuration, using the DDR4 system memory (Top) and the MCDRAM (bottom). Notice the different scales on the y-axes.



(b) *collide* function. Flat configuration, using the DDR4 system memory (Top) and the MCDRAM (bottom).

Figure 2. Energy consumption in nano-joules per lattice site, using different memory data layouts and different number of threads. Each bar represents the Package Energy (bottom), plus the DRAM energy (top). When using the MCDRAM, the latter is just the DRAM idle energy consumption.

processor frequencies in order to highlight interesting trade-offs between performance and energy-efficiency. Starting from these previous results, we then investigate the trade-offs available on the KNL processor, using the LB application described in Sec. 2 as a representative of a wider class of lattice-based simulations.

According to the *Roofline Model* [37], any processor based on the Von Neumann architecture, hosts two different subsystems working together to perform a given computation: a compute subsystem with a given computational performance C *FLOPS/s* and a memory subsystem, providing a bandwidth B *Byte/s* between the processor and memory. The ratio $M_b = C/B$, specific of each hardware architecture, is known as *machine-balance* [38]. On the other side, every computational task performed by an application is made up of a certain number of operations O, operating on D data items to be fetched from and written onto memory. Thus, for any software function, the corresponding ratio $I = O/D$ is referred as the *arithmetic intensity*, *computational intensity* or *operational intensity*.

However, in general, this model tell us that, given a hardware architecture with a machine-balance M_b, the performance of a software function with a computational intensity I, would be limited by the hardware memory subsystem if $I < M_b$, or by the compute subsystem if $I > M_b$. Software optimizations attempting to change I in order to mach the available M_b are indeed a well known practice to maximize applications performance, which often translate also to an increase in energy-efficiency [10].

Default OS frequency governors commonly set the higher available processor frequency, when an application is running, in order to maximize the system M_b (and thus performance), independently from the actual application needs. As an example, a memory-bound application with $I << M_b$ would not appreciate any performance benefit from a higher processor clock, but more energy would be spent. On the other side, changing the processor clock frequency using DVFS, an user may decrease the processor clock to lower the value M_b, in order to match a function specific I. This would not lead to a performance increase neither, but lowering the processor clock (and correspondingly lowering also the processor supply voltage) gives a lower power dissipation (theoretically without impacting performances), and thus an higher energy-efficiency.

The *Roofline Model* has been recently used to study in detail the KNL architecture [39], highlighting the fact that the M_b value is obviously different if we use the MCDRAM or the DDR4 memory. In our case, the KNL 7230 has a $M_{b,MC} \approx \frac{2662}{450} = 5.9$ FLOP/Byte using the MCDRAM and $M_{b,DDR} \approx \frac{2662}{115.2} = 23.1$ FLOP/Byte using the DDR4 main memory.

Concerning the LB application described in Sec. 2, the *propagate* function, is completely memory-bound, while the *collide* function has an *arithmetic intensity* $I_{\text{collide}} \approx 13.3$. This suggests that on this architecture $M_{b,MC} < I_{\text{collide}} < M_{b,DDR}$ and thus the *collide* function, which is commonly compute-bound on most architectures, should become memory-bound when using the DDR4 main memory.

6.1. Function benchmarks

We start benchmarking both the *propagate* and *collide* functions, sweeping all the user configurable frequencies, highlighting the available trade-off between performance and energy-efficiency for each of them and selecting optimal frequencies for both E_S and T_S metrics.

Our benchmark application is instrumented to change the processor clock frequencies from within the application itself. It uses the *acpi_cpufreq* driver of the Linux kernel [2], able to set a specific frequency on each core by calling the *cpufreq_set_frequency()* function. The *Userspace* cpufreq governor has to be loaded in advance, in order to disable the governors managing the dynamic frequency scaling and to be able to manually select a fixed processor frequency.

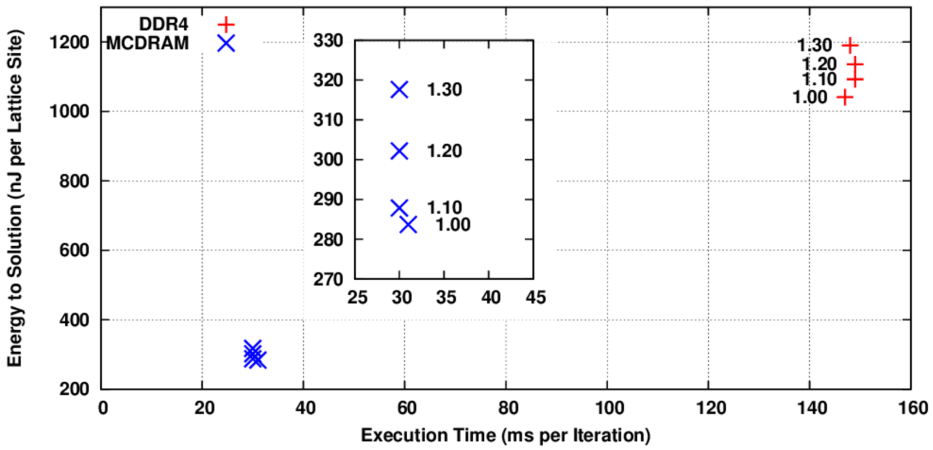
In Figure 3a we show the energy-to-solution E_S metric as a function of the time-to-solution T_S for the *propagate* function using the *CAoSoA* data layout. As expected, being this function completely memory-bound, a decrease in the processor frequency does not lead to an increase in the execution time, but allows to save $\approx 15\%$ of energy, both for the configurations using the external DDR4 memory or the internal MCDRAM.

---

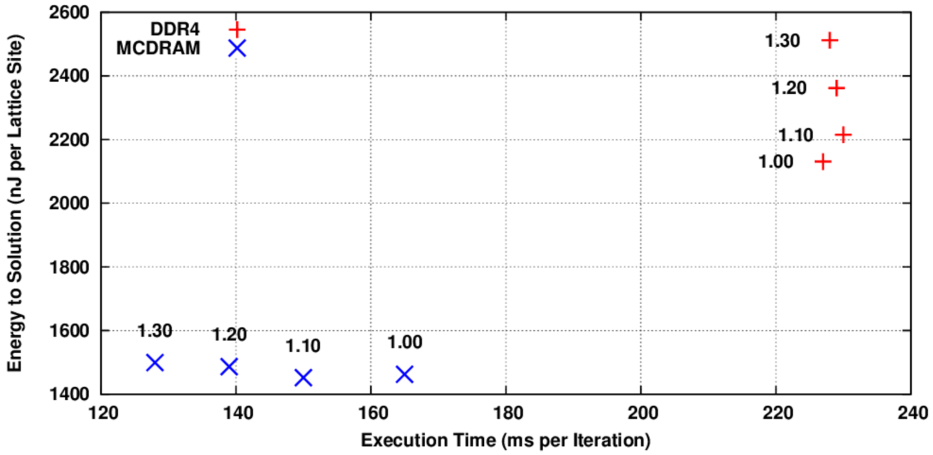
[2] As of Linux Kernel 3.9 the default driver for Intel Sandy Bridge and newer CPUs is *intel_pstate*, we disabled it in order to be able to use the *acpi_cpufreq* driver instead.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 April 2018 doi:10.20944/preprints201804.0074.v1

Peer-reviewed version available at J. Low Power Electron. Appl. 2018, 8, 18; doi:10.3390/jlpea8020018



(a) *propagate* function using respectively the DDR4 (Red) and MCDRAM (Blue) memories.



(b) *collide* function using respectively the DDR4 (Red) and MCDRAM (Blue) memories.

Figure 3. E_S (*energy-to-solution*) versus T_S (*time-to-solution*) for the *propagate* and *collide* functions adopting the *CAoSoA* data layout and using 256 threads. KNL cores frequencies shown in GHz as labels.

In Figure 3b we show the results of the same test for the *collide* function. In this case is interesting to underline that, as predicted by the *Roofline Model*, this function is strongly compute-bound using the MCDRAM memory, while it becomes memory-bound when using the slower DDR4 memory. In the former case, in fact, its T_S is heavily impacted by a processor frequency decrease, while there is a negligible E_S benefit. On the other hand, in the latter case, a $\approx 20\%$ energy-saving can be appreciated with almost no impact on performance.

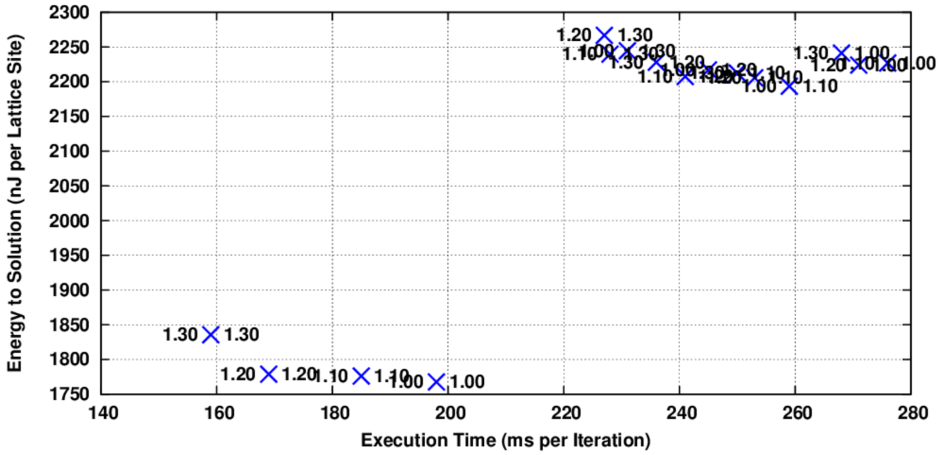
These findings suggest that the optimal frequency for the *propagate* function, as expected, is always the lowest value (i.e., 1.0 GHz), independently from the memory configuration used, either MCDRAM or DDR4. This holds true also for the *collide* function, when using the DDR4 memory, but not anymore when using the MCDRAM. In this latter case a Pareto front is present, where to look for a trade-off between E_S and T_S, although the possible energy-saving is in the order of just $\approx 5\%$, practically suggesting to run at full throttle also from the E_S point of view.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 April 2018 doi:10.20944/preprints201804.0074.v1

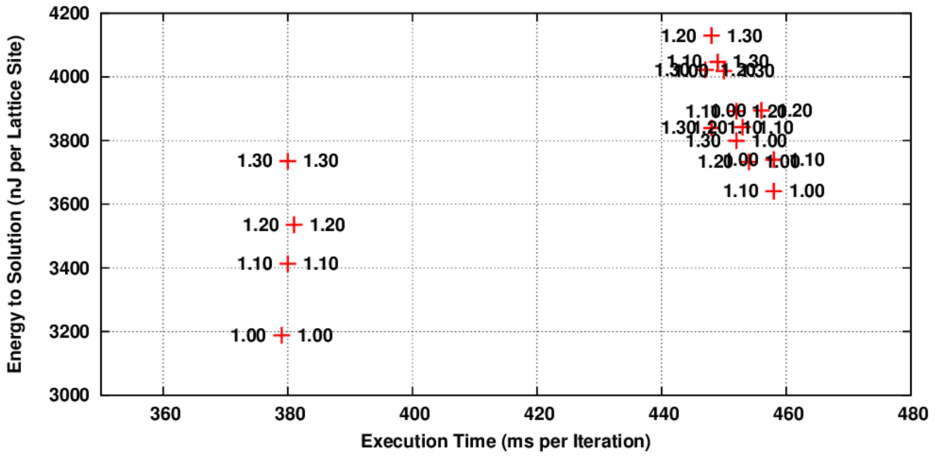
Peer-reviewed version available at J. Low Power Electron. Appl. 2018, 8, 18; doi:10.3390/jlpea8020018

6.2. Full Application Results

Using the results from previous section, we conclude that – at least for the case of our application – the best strategy for computing performance and energy efficiency is to store the lattice onto MCDRAM, and use two different clock frequencies for the two main functions – *propagate* and *collide* – of the application. In particular the processor frequency should be lowered to the minimum value before running the *propagate* kernel, and then increased to the maximum value before executing the *collide* kernel. To test the feasibility and the corresponding benefits of this strategy, we run a real simulation test changing the clock frequency of the KNL processor from the application itself, and try all available clock frequencies for the two functions.



(a) Full simulation storing the lattice in the MCDRAM memory.



(b) Full simulation storing the lattice in the DDR4 memory.

Figure 4. E_S (*energy-to-solution*) versus T_S (*time-to-solution*) for the whole simulation adopting the *CAoSoA* data layout and using 256 threads. KNL cores frequencies for the two functions are shown in GHz; label on the left for *propagate* and label on the right for *collide*.

In Figure 4a we show our results, allocating the lattice on the MCDRAM memory and changing the KNL cores frequency before launching each function. No frequency changes are made for the tests in which both functions are run at the same frequency. We see that the performance of all runs adopting different frequencies for the two functions is badly impacted. This is due to the fact that the latency time associated to a change in core frequency is large, of the order of tens of milliseconds (2 changes for each iteration). This phenomena is

the same already observed for NVIDIA GPUs [10] where the time cost of each clock change has been measured as $\approx$ 10ms, in sharp contrast to Intel Haswell CPUs where it is $\approx 10\mu$s. In Figure 4b we show the results of the same tests allocating the lattice on the DDR4. Similar conclusions can be drawn concerning the time cost of each clock change, but we also see that the whole application behavior is mainly memory-bound when running on the DDR4, while it is compute-bound when running on MCDRAM. This suggest that for relatively small lattice sizes, able to fit in the MCDRAM there is a possible trade-off between performance and energy-efficiency, but at first approximation, an higher frequency would be desiderable. On the contrary, for large lattice sizes, not able to fit in the MCDRAM, almost $\approx 20\%$ of the energy could be saved, lowering the KNL cores frequency to the minimum value, without impacting the performance of the whole application.

7. Conclusion and future works

In this work we have investigated the energy-efficiency of the Intel KNL for Lattice Boltzmann applications, assessing the *energy to solution* for the most relevant compute kernels, i.e., *propagate* and *collide*. Based on our experience in using the KNL, related to our application, and the experimental measures we have done, some concluding remarks are in order:

i) applications previously developed for ordinary x86 multi-core CPUs can be easily ported and run on KNL processors. However performance is strongly related to the level of vectorization and core parallelism that applications are able to exploit;

ii) for LB – and for many others – applications, appropriate data layouts play a relevant role to allow for vectorization and for an efficient use of the memory sub-system, improving both computing and energy efficiency;

iii) if application data fits within the MCDRAM, performance of KNL are very competitive with that of recent GPUs in terms of both computing and energy-efficiency; unfortunately, if this is not the case, computing performance is strongly reduced;

iv) the machine-balance is strongly reduced when using DDR4, but the performance degradation could be compensated by an energy-saving of up to 20% using DVFS to reduce the cores frequency.

In the future, we plan to further investigate the energy-efficiency of the KNL, comparing it also to other recent architectures. We also would like to adopt more handy tools for performance and energy profiling [40], allowing to correlate processor performance counters with performance and energy metrics, fostering finer grained analysis.

Acknowledgments: This work was done in the framework of the COKA, COSA projects of INFN, and the PRIN2015 project of MIUR. We would like to thank CINECA (Italy) for the access to their HPC systems. AG has been supported by the EU Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 642069.

Author Contributions: The authors have jointly contributed to the development of this project and to the writing of this research paper. In particular: E.C. focused on the development and implementation of tools to set the processor clock frequencies and analysis of the corresponding energy consumption; A.G. and SF.S. have contributed to the design and implementation of the different data-structures and to the analysis of computing performances; and R.T. has designed and developed the Lattice Boltzmann algorithm.

The authors declare no conflict of interest.

References

Bibliography

1. Ge, R.; Feng, X.; Song, S.; Chang, H.C.; Li, D.; Cameron, K.W. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems* **2010**, *21*, 658–671.
2. Attig, N.; Gibbon, P.; Lippert, T. Trends in supercomputing: The European path to exascale. *Computer Physics Communications* **2011**, *182*, 2041 – 2046.
3. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Early experience on using Knights Landing processors for Lattice Boltzmann applications. Parallel Processing and Applied Mathematics: 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017, 2018, Vol. 1077, *Lecture Notes in Computer Science*, pp. 1–12.

4. Bernard, C.; Christ, N.; Gottlieb, S.; Jansen, K.; Kenway, R.; Lippert, T.; Lüscher, M.; Mackenzie, P.; Niedermayer, F.; Sharpe, S.; Tripiccione, R.; Ukawa, A.; Wittig, H. Panel discussion on the cost of dynamical quark simulations. *Nuclear Physics B - Proceedings Supplements* **2002**, *106*, 199 – 205.
5. Bilardi, G.; Pietracaprina, A.; Pucci, G.; Schifano, F.; Tripiccione, R. The potential of on-chip multiprocessing for QCD machines. *Lecture Notes in Computer Science* **2005**, *3769*, 386–397.
6. Bonati, C.; Calore, E.; Coscetti, S.; D'Elia, M.; Mesiti, M.; Negro, F.; Schifano, S.F.; Tripiccione, R. Development of scientific software for HPC architectures using OpenACC: the case of LQCD. The 2015 International Workshop on Software Engineering for High Performance Computing in Science (SE4HPCS), 2015, ICSE Companion Proceedings, pp. 9–15.
7. Bonati, C.; Coscetti, S.; D'Elia, M.; Mesiti, M.; Negro, F.; Calore, E.; Schifano, S.F.; Silvi, G.; Tripiccione, R. Design and optimization of a portable LQCD Monte Carlo code using OpenACC. *International Journal of Modern Physics C* **2017**, *28*.
8. Bonati, C.; Calore, E.; D'Elia, M.; Mesiti, M.; Negro, F.; Sanfilippo, F.; Schifano, S.; Silvi, G.; Tripiccione, R. Portable multi-node LQCD Monte Carlo simulations using OpenACC. *International Journal of Modern Physics C* **2018**, *29*.
9. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Energy-efficiency evaluation of Intel KNL for HPC workloads. Parallel Computing is Everywhere, 2018, Vol. 32, *Advances in Parallel Computing*, pp. 733–742.
10. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications. *Concurrency and Computation: Practice and Experience* **2017**, *29*, 1–19.
11. Succi, S. *The Lattice-Boltzmann Equation*; Oxford university press, Oxford, 2001.
12. Biferale, L.; Mantovani, F.; Sbragaglia, M.; Scagliarini, A.; Toschi, F.; Tripiccione, R. Second-order closure in stratified turbulence: Simulations and modeling of bulk and entrainment regions. *Physical Review E* **2011**, *84*, 016305.
13. Biferale, L.; Mantovani, F.; Pivanti, M.; Sbragaglia, M.; Scagliarini, A.; Schifano, S.F.; Toschi, F.; Tripiccione, R. Lattice Boltzmann fluid-dynamics on the QPACE supercomputer. *Procedia Computer Science* **2010**, *1*, 1075–1082. ICCS 2010.
14. Sbragaglia, M.; Benzi, R.; Biferale, L.; Chen, H.; Shan, X.; Succi, S. Lattice Boltzmann method with self-consistent thermo-hydrodynamic equilibria. *Journal of Fluid Mechanics* **2009**, *628*, 299–309.
15. Scagliarini, A.; Biferale, L.; Sbragaglia, M.; Sugiyama, K.; Toschi, F. Lattice Boltzmann methods for thermal flows: Continuum limit and applications to compressible Rayleigh–Taylor systems. *Physics of Fluids (1994-present)* **2010**, *22*, 055101.
16. Biferale, L.; Mantovani, F.; Sbragaglia, M.; Scagliarini, A.; Toschi, F.; Tripiccione, R. Reactive Rayleigh-Taylor systems: Front propagation and non-stationarity. *EPL* **2011**, *94*, 54004.
17. Biferale, L.; Mantovani, F.; Pivanti, M.; Pozzati, F.; Sbragaglia, M.; Scagliarini, A.; Schifano, S.F.; Toschi, F.; Tripiccione, R. A Multi-GPU Implementation of a D2Q37 Lattice Boltzmann Code. In *Parallel Processing and Applied Mathematics: 9th International Conference, PPAM 2011, Torun, Poland, September 11-14, 2011. Revised Selected Papers, Part I*; Lecture Notes in Computer Science, Springer Berlin Heidelberg: Berlin, Heidelberg, 2012; pp. 640–650.
18. Calore, E.; Schifano, S.F.; Tripiccione, R. On Portability, Performance and Scalability of an MPI OpenCL Lattice Boltzmann Code. In *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II*; Lecture Notes in Computer Science, Springer International Publishing: Cham, 2014; pp. 438–449.
19. Calore, E.; Schifano, S.F.; Tripiccione, R. Energy-performance tradeoffs for HPC applications on low power processors. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2015**, *9523*, 737–748.
20. Calore, E.; Gabbana, A.; Kraus, J.; Schifano, S.F.; Tripiccione, R. Performance and portability of accelerated lattice Boltzmann applications with OpenACC. *Concurrency and Computation: Practice and Experience* **2016**, *28*, 3485–3502.
21. Calore, E.; Gabbana, A.; Kraus, J.; Pellegrini, E.; Schifano, S.F.; Tripiccione, R. Massively parallel lattice-Boltzmann codes on large GPU clusters. *Parallel Computing* **2016**, *58*, 1 – 24.
22. Mantovani, F.; Pivanti, M.; Schifano, S.F.; Tripiccione, R. Performance issues on many-core processors: A D2Q37 Lattice Boltzmann scheme as a test-case. *Computers & Fluids* **2013**, *88*, 743 – 752.

23. Crimi, G.; Mantovani, F.; Pivanti, M.; Schifano, S.F.; Tripiccione, R. Early Experience on Porting and Running a Lattice Boltzmann Code on the Xeon-phi Co-Processor. *Procedia Computer Science* **2013**, *18*, 551–560.
24. Calore, E.; Demo, N.; Schifano, S.F.; Tripiccione, R. Experience on Vectorizing Lattice Boltzmann Kernels for Multi- and Many-Core Architectures. In *Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part I*; Lecture Notes in Computer Science, Springer International Publishing: Cham, 2016; pp. 53–62.
25. McCalpin, J.D. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* **1995**, pp. 19–25.
26. Colfax. Clustering modes in Knights Landing processors, 2016.
27. Colfax. MCDRAM as high-bandwidth memory (HBM) in Knights Landing processors: Developers guide, 2016.
28. Sodani, A.; Gramunt, R.; Corbal, J.; Kim, H.S.; Vinod, K.; Chinthamani, S.; Hutsell, S.; Agarwal, R.; Liu, Y.C. Knights landing: Second-generation Intel Xeon Phi product. *IEEE Micro* **2016**, *36*, 34–46.
29. Dongarra, J.; London, K.; Moore, S.; Mucci, P.; Terpstra, D. Using PAPI for hardware performance monitoring on Linux systems. Conference on Linux Clusters: The HPC Revolution. Linux Clusters Institute, 2001, Vol. 5.
30. Weaver, V.; Johnson, M.; Kasichayanula, K.; Ralph, J.; Luszczek, P.; Terpstra, D.; Moore, S. Measuring Energy and Power with PAPI. Parallel Processing Workshops (ICPPW), 2012 41st International Conference on, 2012, pp. 262–268.
31. Hackenberg, D.; Schone, R.; Ilsche, T.; Molka, D.; Schuchart, J.; Geyer, R. An Energy Efficiency Feature Survey of the Intel Haswell Processor. Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International, 2015, pp. 896–904.
32. Desrochers, S.; Paradis, C.; Weaver, V.M. A Validation of DRAM RAPL Power Measurements. Proceedings of the Second International Symposium on Memory Systems, 2016, MEMSYS '16, pp. 455–470.
33. Calore, E.; Gabbana, A.; Schifano, S.F.; Tripiccione, R. Optimization of lattice Boltzmann simulations on heterogeneous computers. *The International Journal of High Performance Computing Applications* **2017**, pp. 1–16.
34. Etinski, M.; Corbalán, J.; Labarta, J.; Valero, M. Understanding the future of energy-performance trade-off via DVFS in HPC environments. *Journal of Parallel and Distributed Computing* **2012**, *72*, 579–590.
35. Haidar, A.; Jagode, H.; YarKhan, A.; Vaccaro, P.; Tomov, S.; Dongarra, J. Power-aware computing: Measurement, control, and performance analysis for Intel Xeon Phi. 2017 IEEE High Performance Extreme Computing Conference (HPEC), 2017, pp. 1–7.
36. Haidar, A.; Jagode, H.; YarKhan, A.; Vaccaro, P.; Tomov, S.; Dongarra, J. Power-aware HPC on Intel Xeon Phi KNL processors, 2017.
37. Williams, S.; Waterman, A.; Patterson, D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **2009**, *52*, 65–76.
38. McCalpin, J.D. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter* **1995**.
39. Doerfler, D.; Deslippe, J.; Williams, S.; Oliker, L.; Cook, B.; Kurth, T.; Lobet, M.; Malas, T.; Vay, J.L.; Vincenti, H. Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor. High Performance Computing; Taufer, M.; Mohr, B.; Kunkel, J.M., Eds., 2016, pp. 339–353.
40. Mantovani, F.; Calore, E. Multi-Node Advanced Performance and Power Analysis with Paraver. Parallel Computing is Everywhere, 2018, Vol. 32, *Advances in Parallel Computing*, pp. 723–732.