A Comparative Study of Application Performance and Scalability on the Intel Knights Landing Processor

Carlos Rosales(✉), John Cazes, Kent Milfeld, Antonio Gómez-Iglesias, Lars Koesterke, Lei Huang, and Jerome Vienne

Texas Advanced Computing Center, The University of Texas at Austin, Austin, TX, USA
{carlos,cazes,milfeld,agomez,lars,huang,viennej}@tacc.utexas.edu

Abstract. Intel Knights Landing represents a qualitative change in the Many Integrated Core architecture. It represents a self-hosted option and includes a high speed integrated memory together with a two dimensional mesh used to interconnect the cores. This leads to a number of possible runtime configurations with different characteristics and implications in the performance of applications. This paper presents a study of the performance differences observed when using the three MCDRAM configurations available in combination with the three possible memory access or cluster modes. We analyze the effects that memory affinity and process pinning have on different applications. The Mantevo suite of mini applications and NAS Parallel Benchmarks are used to analyze the behavior of very different application kernels, from molecular dynamics to CFD mini-applications. Two full applications, the Weather Research and Forecast (WRF) application and a Lattice Boltzman Suite (LBS3D) are also analyzed in detail to complete the study and present scalability results of a variety of applications.

Keywords: KNL · MCDRAM · Scalability · MIC

1 Introduction

Over recent years one of the design criteria in HPC systems has been power efficiency. Efforts to save power have been applied to all levels, from megawatt savings in power conversion and cooling at the center level, to picojoule savings in logical units and data transfers [17]. The new designs will certainly benefit the effective carbon footprint of data centers; but moreover, these changes will benefit the efforts to create an HPC exaflop machine with reasonable power requirements.

A significant surge in Floating Point operation efficiency was realized when GPUs, which already had a significant single precision performance, included a CUDA paradigm [11] and microprocessor features for the HPC community to use these GPGPUs as floating point accelerator devices.

M. Taufer et al. (Eds.): ISC High Performance Workshops 2016, LNCS 9945, pp. 307–318, 2016.
DOI: 10.1007/978-3-319-46079-6_22

Meanwhile, during the same period, core replication became the mode for increasing workload capacity, as power restraints restricted further increases in clock frequencies. Further parallelism was achieved by increasing the number of Vector Processing Units (VPU) per core, widening the registers from 128b to 512b, and making them FMA capable.

Even with all the efforts in core and VPU units (also known as SIMD units), Intel also directed their efforts into accelerator capabilities in a series of programs (Larrabee [12]) that culminated in the Many Integrated Core (MIC) Architecture [4]. The Phi product line of these many-core systems began with the Knights Corner (KNC) as a coprocessor board attached to a host through a PCI-e bus. The coprocessor interacts with the host through the PCI-e bus, similar to the way a GPGPU does.

Because of the characteristics previously described, there are currently many accelerated systems in the Top 500[1]. These systems present nodes with at least one GPGPU or one MIC, although it is possible to find systems in which each node has two or more accelerators. There are also systems that present both types of accelerators in different nodes.

Most HPC programmers realize that a major bottleneck of accelerated computing is in the speed of the PCI-e bus; and there is a limited memory capacity on the device since only GDDR (Graphics DDR) memory is used. While some applications perform well within these constrains, for other applications it is difficult to shoehorn their algorithms into a remote device with limited memory.

The 2^{nd} generation Intel® Xeon Phi™ processor, code named Knights Landing (KNL), has architectural features that are designed to solve the shortcoming mentioned directly above, and address the power efficiency concerns at a microprocessor level [14].

The microprocessor architecture has features (southside bus, etc.) that allow the processor to run as a bonafide stand-alone system. Hence, there are no execution offloads through a PCI-e bus, as is required when using multiple KNCs. Also, chips with Omni-Path [2], the new Intel fabric, can bypass the PCI-e bus for external communication. There are two different types of memory in KNL: MCDRAM and DDR4. The on-module MCDRAM [14] provides the high-speed memory that accelerated applications have become accustomed to; and the usual DDR4 memory provides the capacity storage many HPC applications require.

KNL cores are organized as tiles, where each tile is comprised of two cores. While L1 cache is implemented at the core level, L2 cache is shared at the tile level. Tiles are interconnected using a mesh (as opposed to the bidirectional ring in the KNC). All the cores on the chip are cache coherent, so that the tile with a specific data can supply that data to another tile in the chip. This mesh can be clustered to achieve a higher performance for specific memory access patterns in applications. The three modes of operation are: All-to-All, Quadrant and Sub-NUMA clustering [14].

There will always be an ongoing effort in the HPC community to adapt to the new computing technologies that provide significant opportunities to

[1] http://www.top500.org/.

compute efficiently and faster. As stated, KNL systems provide new technologies in memory access, communication, and SIMD execution. In this article we explore these technologies with applications and benchmarks, and report programming concepts that will be useful in adapting applications to these new features.

The rest of the paper is organized as follows: Sect. 2 introduces the different applications that we use for our experiments as well as different configurations of the hardware, while the results of those experiments are presented in Sect. 3. Finally, Sect. 4 summarizes the paper and presents a set of ideas that will be explored in the future.

2 Background

We have selected a set of applications and miniapps to study the different configurations of Intel KNL regarding MCDRAM and cluster modes. The miniapps are representative of popular applications in HPC environments.

2.1 Mantevo

The Mantevo suite [5] provides a set of application proxies or miniapps that can be used to measure the performance of hardware. These are self-contained, stand-alone applications. They represent some of the most common scenarios in scientific computing and include numerical kernels that focus on specific aspects of the hardware. We focus on the following miniapps for the experiments presented in this paper:

1. **MiniFE**: a finite elements application. It solves a nonlinear system of equations. Like most of the codes that solve these functions, a large portion of the time is spent in a conjugate gradient solver. While it can be configured to study the repercussions of, for example, load imbalance in the execution, we will focus on well-balanced test cases. It is a memory-bound application, which makes it an optimal candidate to study the impact of the different memory configurations previously introduced. The operations performed by the application greatly depend on memory throughput and, when many cores are used within one processor, it often leads to CPU stalls. The application has received a lot of attention in the past and it is possible to find diverse implementations of MiniFE with different levels of optimization. It is a C++ code, parallelized with MPI and OpenMP. We use the reference OpenMP version.
2. **MiniMD**: molecular dynamics code. This is a small version of the well-known code LAMMPS. It implements spatial decomposition, where each processor works on subsets of the simulation box. MiniMD computes atoms movements in a 3D space using the Lennard-Jones pair interaction. It follows a stencil communication pattern where neighbors exchange information about atoms in

boundary regions. Because of these characteristics, it provides good weak scaling. It also presents different implementations in C++ (MPI+OpenMP, OpenSHMEM,...), and we will focus on the MPI+OpenMP version in this paper. A single MPI task was used for all the runs.

2.2 NAS Parallel Benchmarks

The well-documented NAS Parallel Benchmarks (NPB) [1,8,16] is a suite of parallel workloads designed to evaluate performance of various hardware and software components of a parallel computing system. These benchmarks span different problem sizes, called classes in NPB terminology and in this paper we use class C, which is standard for the analysis of single-node systems. Most of the NAS benchmarks are computational kernels. **IS** performs sorting of integer keys using a linear time Integer Sorting algorithm based on computation of the key histogram. **EP** evaluates an integral by means of pseudorandom trials. **FT** contains the computational kernel of a 3-D Fast Fourier Transform (FFT). **CG** uses a Conjugate Gradient method to compute approximations to the smallest eigenvalues of a sparse unstructured matrix. **MG** uses a V-cycle Multi Grid method to compute the solution of the 3-D scalar Poisson equation. **LU**, **SP**, and **BT** are simulated CFD mini-applications that solve the discretized compressible Navier-Stokes equations. **BT** and **SP** both apply variations of the Alternating Direction Implicit (ADI) approximate factorization technique to decouple solution in the x, y, and z-coordinate directions. The resulting systems are 5×5 block-tridiagonal and scalar pentadiagonal, respectively, which can be solved independently. **LU** applies the symmetric successive over-relaxation (SSOR) technique to an approximate factorization of the discretization matrix into block-lower and block-upper triangular matrices. **UA** evaluates unstructured computation, parallel I/O, and data movement using unstructured adaptive mesh with dynamic and irregular memory accesses.

2.3 WRF

The Weather Research and Forecasting(WRF) Model [13] is a widely used numerical weather prediction system used for both research and operational forecasts. WRF is primarily a Fortran code implemented using MPI and OpenMP for distributed computing. The problem space on each process is divided into tiles that are processed by OpenMP threads. Ideally, the best performance is achieved when the size of the tile (in terms of the problem in WRF) fits into the smallest cache. Having multiple application tiles allows WRF to obtain high levels of memory bandwidth utilization.

A substantial effort was made to optimize WRF for the first generation Xeon Phi, Knights Corner [7]. The current version of WRF, 3.7.1, supports a configuration option for the KNC. This configuration was modified to compile the KNL instruction set using the `-xMIC-AVX512` option rather than the `-mmic` option. The source code was not modified for this study.

2.4 LBS3D

LBS3D is a multiphase Lattice Boltzmann Code based on the Free Energy method of Zheng, Shu and Chew [18]. This code simulates the flow of two immiscible, isothermal, incompressible fluids with great spatial and temporal detail. For details on the model we refer the interested reader to [9,15]. LBS3D is an optimized implementation of this model, originally developed for execution in the first generation Intel Xeon Phi [10]. While both OpenMP and hybrid MPI+OpenMP versions of the code are available, the results reported in this work are for the OpenMP version only.

2.5 MCDRAM Modes

The MCDRAM and the DDR4 can be configured in three different modes: Flat, Cache and Hybrid. This section describes each of them briefly.

- **Flat Mode** MCDRAM memory appears to the programmer as a continuation of the main memory. Allocation in the 16 GB MCDRAM area or the DDR4 area is determined by NUMA controls or specific allocation calls with the `memkind` library [3]. This mode should be optimal for applications with high memory bandwidth requirements but moderate memory footprint.
- **Cache Mode** MCDRAM is treated as an effective Level 3 cache between the KNL tiles and the main DDR4 memory on the node. Memory allocation and transfers are controlled automatically by the OS kernel. This mode should be optimal for applications with very large memory footprint, significant memory bandwidth requirements, and a regular memory access pattern.
- **Hybrid Mode** MCDRAM can be statically divided into 25/50/75 % blocks to be used as cache or flat memory. This mode may be best for advanced users wishing to fully optimize their code or workflow.

2.6 Memory Access Modes

On top of three basic configuration modes for MCDRAM, KNL offers multiple ways to group coherency across the many cores/tiles in the processor. The following access modes, among others that differ only slightly from these, are available:

- **All-to-All** Cache tag directory is distributed across all tiles.
- **Quadrant** Cache tag directory located in the same quadrant as the corresponding memory. Should improve latency with respect to All-to-All mode when most accesses are local.
- **Sub-NUMA Cluster 2/4** Each half or quadrant is exposed as a separate NUMA node by subdividing the tiles into clusters. This configuration may be of high interest to users of hybrid codes trying to balance the number of MPI tasks and OpenMP threads used during execution. Currently this seems to be available only for KNL processors with 36 active tiles, and we have been unable to test this particular configuration.

3 Evaluations

All the results presented in this work were obtained on a Knights Landing pre-production system, B0 stepping, 1.30 GHz, 64 cores (32 tiles), 16 GB MCDRAM, 96 GB DDR4 (16×6 DDR4 2133 DIMMs), run with multiple clustering and memory modes. At the time of submitting this paper we have been able to complete experiments with the following memory configurations for KNL: Flat Mode (All-to-All and Quadrant), Cache (All-to-All). In the case of the Flat Mode we used NUMA memory policy to determine where memory should reside (MCDRAM or main DDR4 memory). For example to run purely on the MCDRAM we used `numactl --membind=1 ./executable` and to run purely on the main memory we used `numactl --membind=0 ./executable`. No such control is available in Cache Mode, of course, since the MCDRAM is not exposed as a NUMA node in that case.

3.1 Memory Access Scaling

To begin, we created a simple saxpy-loop program to mimic the triad benchmark in STREAM [6], so that we could quickly determine a profile showing how the memory bandwidth scales with the thread count. The loop, consisting of 100,000,000 iterations with double precision data, is repeated 200 times (after data was initialized), and compiled with only -O3 and -xMIC-AVX2 optimizations. We present these results first, so that the reader has a clear picture of the bandwidth capability at different thread counts.

Four types of memory access are shown in Fig. 1. On a node configured with FLAT mode memory and All-to-All Clustering, three experiments were performed: scaling using DDR4 memory access, by simply invoking the executable; scaling using MCDRAM, by accessing MCDRAM through a NUMA command (MC-numa) at execution; and scaling using MCDRAM, through a memkind library call to hbw_malloc within the code (MC-hbw). The fourth experiment used a node configured with Cache mode memory and All-to-All clustering (MC-cache).

The simple-saxpy results show that a maximum DDR4 bandwidth of 82 GB/s is reached with 14 threads, while a maximum MCDRAM bandwidth of 419 GB/s is reached with 64 thread (for MC-numa). These profiles match the triad STREAM benchmarks that users can derive from the `micprun -k stream` command, preinstalled on the KNL. The micprun scan reports STREAM triad maximums of 82 GB/s and 428 GB/s for DDR4 and MCDRAM, respectively.

Note, the scaling profiles of MC-numa, MC-cache, and MC-hbw are all the same, with the same performance ordering throughout the whole range. NUMA controlled bandwidths are the highest, followed by cache accesses; and accesses through hbw_malloc have the worst performance. The difference between MC-numa and MC-hbw is about 12 % for the first and last 10 thread counts, and 7 % throughout the middle of the curve. While it does seem reasonable that MCDRAM access through cache (MC-cache) may have a performance hit (relative to a non-cached mode), one would expect no difference between the MC-numa and MC-hbw experiments.

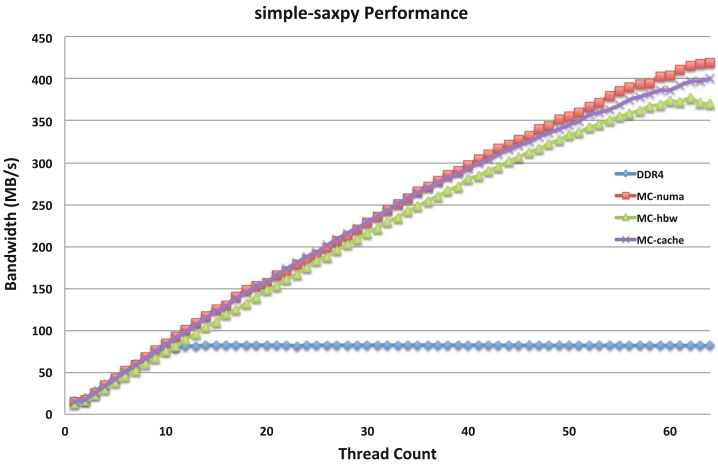

Fig. 1. Bandwidth scaling for DDR4 and MCDRAM memory.

3.2 Memory Configuration Effect on MCDRAM Contained Workloads

This section describes the effect of different memory configuration settings for workloads that can be fitted inside the MCDRAM memory footprint (MiniFE: problem size $256 \times 256 \times 256$ (approximately 6 GB footprint); MiniMD: 500K atoms and 1K iterations (300 MB); WRF: a standard CONUS 12 km benchmark case was used; LBS3D: size $256 \times 256 \times 256$, with a footprint of approximately 6 GB; NAS Class C with a memory footprint of approximately 6 GB). All results correspond to the average of multiple runs with no significant deviation in runtimes across the sample.

Figures 2 and 3 show the effect of memory configuration on all the applications studied. Performance has been normalized to that of running in the slower of the measured modes for each application individually, in order to be able to show all results in a single graph. Notice how MiniMD, which is highly insensitive to memory bandwidth, sees little difference from the change in configuration, while LBS3D, MiniFE and NAS BT, EP, FT MG and SP see an remarkable speedup when running inside MCDRAM. It can be explained by the fact that these applications are sensitive to the bandwidth.

While the performance of WRF is strongly correlated with memory bandwidth, it does have more computational overhead than LBS3D and MiniFE. Hence, the performance improvements for WRF are slightly less than those for the simpler LSB3D and MiniFE codes.

Keep in mind that all the workloads fit in MCDRAM in this case, which explains how closely Cache and Flat/MCDRAM modes are in all cases. In the Fig. 2, `A2A` refers to All-to-All mode, while `Quad` refers to Quadrant mode. NAS benchmarks presented in Fig. 3 were only executed in All-to-All mode.

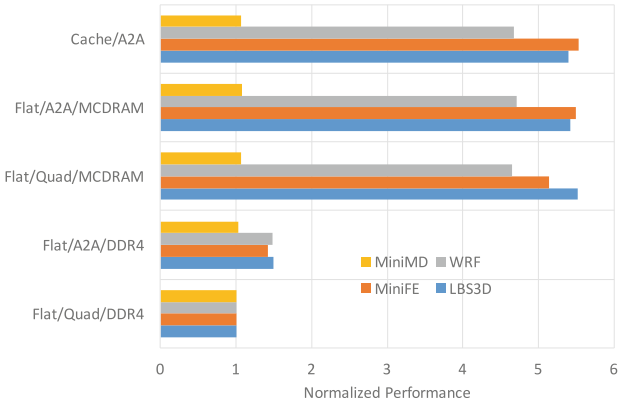

Fig. 2. Comparative effect of memory configuration on MCDRAM-contained workloads.

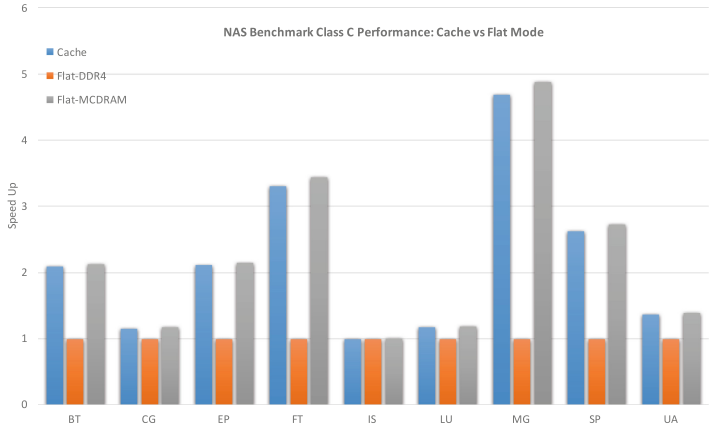

Fig. 3. Comparative effect of memory configuration on MCDRAM-contained workloads using NAS

3.3 Memory Configuration Effect on Non-MCDRAM Contained Workloads

We have compared the performance of several large data sets, from 6 GB to 48 GB memory footprints, when running in Cache Mode, and when running in Flat Mode and allocating to main DDR4 memory. Figure 4 shows that there is a significant improvement in performance across all executions, with a minimum improvement of almost 19 % for both LBS3D and miniFE. As expected, the benefit of using cache mode decreases as the workload grows beyond the cache size. The fact that an improvement of nearly 19 % is still present for workloads with memory footprints that triple the MCDRAM cache size is a good indicator of the benefit of running in this configuration mode.

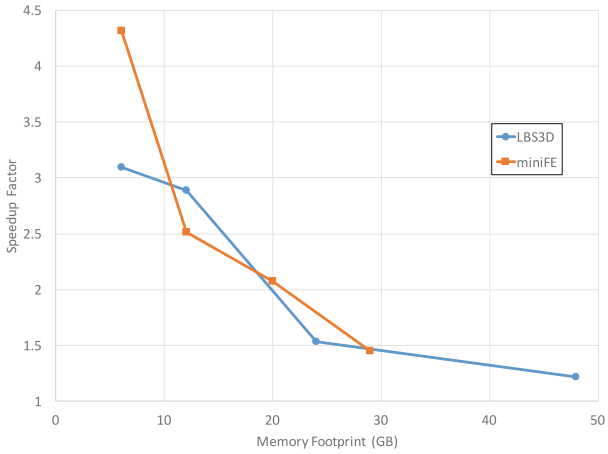

Fig. 4. Effect of Cache Mode for non-MCDRAM contained workload (LBS3D and MiniFE).

Our tests show that the benefit from using cache mode is not independent of the type of memory access and the number of threads used, which is expected. More work will be necessary in this area to fully characterize the benefits of using the cache mode, but the results obtained so far are encouraging.

3.4 Scalability and Memory Configuration

Performance for a fixed workload was measured for each of the applications in each of the previously mentioned memory configuration modes.

Figure 5 shows excellent scalability for MiniFE when using MCDRAM, with the code scaling very strongly throughout the range of 1 thread per core (up to 64 threads), and then speedup continues at a smaller rate beyond that. When allocating to DDR4 memory the scalability stalls between 16 and 32 threads, with the All-to-All clustering outperforming Quadrant cluster asymptotically. This is because the available memory bandwidth to main memory has been exhausted.

LBS3D behaves in a similar manner, as shown in Fig. 6, with very strong scalability up to 128 threads. The main difference between the results for LBS3D and MiniFE is that after 128 threads LBS3D actually starts to slow down even when running in MCDRAM. This is most likely due to the sheer number of streams in flight, which lowers prefetcher efficiency and increases the likelihood of conflicts. The LBS3D performance also seems to stall earlier, around 16 threads, when allocation is purposefully set in the main system DDR4 memory rather than the MCDRAM.

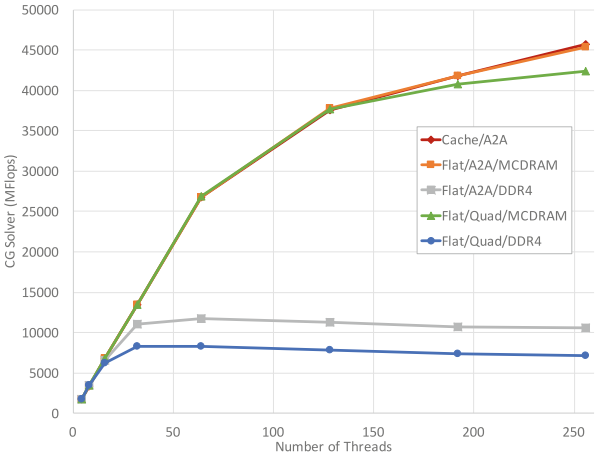

Fig. 5. Scalability of MiniFE on a single KNL processor using different memory configurations.

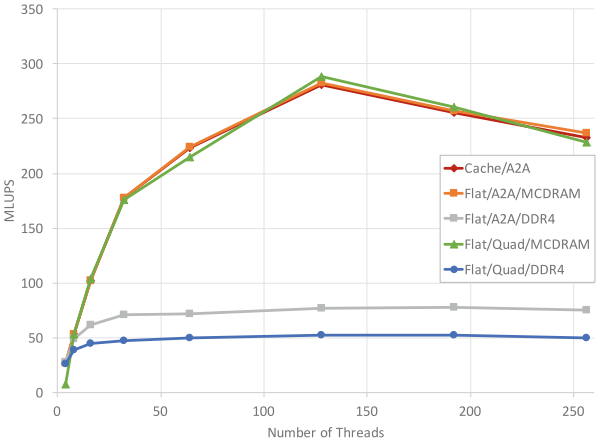

Fig. 6. Scalability of LBS3D on a single KNL processor using different memory configurations.

In the case of MiniMD we observe relatively less scalability all the way through the thread range, with no significant changes between different memory configuration modes (see Fig. 7). The similarity of results from different configurations is expected, since the code does not show a strong dependence on memory bandwidth as is typical for Molecular Dynamics codes. The overall scalability seems weak, but this could be a case where additional tests using multiple MPI tasks are required in order to obtain a higher performance. That work is currently being performed.

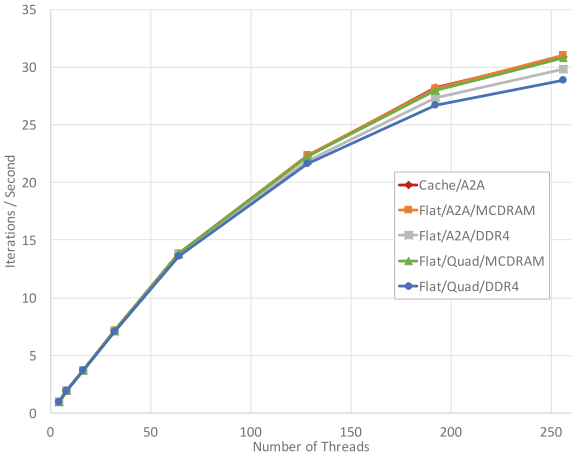

Fig. 7. Scalability of MiniMD on a single KNL processor using different memory configurations.

4 Conclusion and Outlook

This paper has presented a set of studies of the performance of representative applications and miniapps on Intel KNL.

The results show that Cache Mode operates very efficiently for MCDRAM-contained workloads, and that memory-bandwidth bound applications see a performance improvement commensurate with the bandwidth ratio between MCDRAM and main DDR4 memory. Initial scalability results show promise for all the applications considered.

As future work we consider detailed studies of the effect of Cache Mode in workloads that do not fit inside MCDRAM. We also plan on continuing these studies to further understand the implications of pining the different processes/threads to cores.

References

1. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, D., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The NAS parallel benchmarks. Int. J. Supercomputer Appl. **5**(3), 63–73 (1991)
2. Birrittella, M.S., Debbage, M., Huggahalli, R., Kunz, J., Lovett, T., Rimmer, T., Underwood, K.D., Zak, R.C.: Intel® omni-path architecture: enabling scalable, high performance fabrics. In: Hot Interconnects, pp. 1–9. IEEE (2015)
3. Cantalupo, C., Venkatesan, V., Hammond, J.R., Czuryło, K., Hammond, S.: User extensible heap manager for heterogeneous memory platforms and mixed memory policies (2015)

4. Duran, A., Klemm, M.: The Intel many integrated core architecture. In: 2012 International Conference on High Performance Computing and Simulation (HPCS), pp. 365–366, July 2012
5. Heroux, M.A., Doerfler, D.W., Crozier, P.S., Willenbring, J.M., Edwards, H.C., Williams, A., Rajan, M., Keiter, E.R., Thornquist, H.K., Numrich, R.W.: Improving performance via mini-applications. Technical report SAND2009-5574, Sandia National Laboratories, 3 (2009)
6. McCalpin, J.: Stream benchmark (1995). www.cs.virginia.edu/stream/ref.html#what
7. Michalakes, J.: Optimizing weather models for Intel Xeon Phi. Intel Theater Presentation SC 2013 (2013)
8. NASA Advanced Supercomputing Division: NAS parallel benchmarks (2016). http://www.nas.nasa.gov/publications/npb.html. Accessed Jun 2016
9. Nourgaliev, R.R., Dinh, T.N., Theofanous, T., Joseph, D.: The lattice Boltzmann equation method: theoretical interpretation, numerics and implications. Int. J. Multiph. Flow **29**(1), 117–169 (2003)
10. Rosales, C.: Porting to the Intel Xeon Phi: opportunities and challenges. In: Extreme Scaling Workshop (XSW 2013), pp. 1–7. IEEE (2013)
11. Sanders, J., Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents. Addison-Wesley Professional, Reading (2010)
12. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P.: Larrabee: A many-core x86 architecture for visual computing. In: ACM SIGGRAPH 2008 Papers, SIGGRAPH 2008, pp. 18:1–18:15. ACM, New York (2008). http://doi.acm.org/10.1145/1399504.1360617
13. Skamarock, W.C., Klemp, J.B., Dudhia, J., Gill, D.O., Barker, M., Duda, K.G., Huang, X.Y., Wang, W., Powers, J.G.: A description of the advanced research WRF version 3. Technical report, National Center for Atmospheric Research (2008)
14. 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 **36**(2), 34–46 (2016)
15. Succi, S.: The Lattice Boltzmann Equation: For Fluid Dynamics and Beyond. Oxford University Press, Oxford (2001)
16. Wong, F.C., Martin, R.P., Arpaci-Dusseau, R.H., Culler, D.E.: Architectural Requirements and scalability of the NAS parallel benchmarks. In: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing (1999)
17. Wong, P.C., Shen, H.W., Johnson, C.R., Chen, C., Ross, R.B.: The top 10 challenges in extreme-scale visual analytics. IEEE Comput. Graph. Appl. **32**(4), 63 (2012)
18. Zheng, H., Shu, C., Chew, Y.T.: A lattice Boltzmann model for multiphase flows with large density ratio. J. Comput. Phys. **218**(1), 353–371 (2006)